Architecture for the Automated Management of Data Center IT Infrastructure

Habilitationsschrift im Fach Informatik
an der Fakultät für Informatik der
Ludwig-Maximilians-Universität München

von

Dr.-Ing. Sven Graupner

Tag der Einreichung: 7. Juni 2010.

Fachmentorat:

Prof. Dr. Heinz-Gerd Hegering, Ludwig-Maximilians-Universität München
Prof. Dr. Claudia Linnhoff-Popien, Ludwig-Maximilians-Universität München
Prof. Dr. Martin Wirsing, Ludwig-Maximilians Universität-München
Prof. Dr. Jacques Sauvé, Universidade Federal de Campina Grande, Brazil

Table of Contents

Chapter 1

Introduction

The efficient management of data centers is critical in today's information-centered world. Automation has proven to increase efficiency in many industries. Although automation has made significant progress in those industries, the management of data centers still occurs mostly manually today. As a consequence, data center management – as part of overall IT management – is perceived as slow, inflexible and cost intensive. It is seen as an obstacle to innovation and business agility in enterprises, which are relying on effective IT services and their efficient management.

The desire for more agile IT environments, the need to address the increasing scales of data centers and the continuing pressure to reduce cost are consequently major drivers in the IT industry today. Automation is one path to addressing these goals. Automation means replacing human labor with systems performing tasks faster, more reliably and at a lower cost [Jac09].

Like in other industries, not all tasks in IT management are suitable for automation. Automation needs to aim at routine and repetitive tasks, which are often found at the lower layers of IT infrastructure management in data centers.

IT management automation has proven to be difficult in the past. Major obstacles have been the fragmentation of management, both in management systems and in management processes performed by people; the lack of integration of management systems; and the lack of comprehensive and consistent information models providing the appropriate abstractions upon which algorithms could make decisions and perform management tasks. IT management automation solutions built today often fall short of expectations due to these factors [Kell06].

In contrast, the discipline of operating systems has led to a class of software systems that has demonstrated the ability to manage applications and components of a machine environment fully transparently and automatically.

Similarities can be observed between the environment of a machine and the environment of a data center. Both environments include physical components for performing computations, store data and facilitate communication. Both environments operate applications on their components. Management is essential in both environments for executing applications in a controlled and coordinated manner.

Operating systems have developed principles, techniques and technologies over the past 50 years, which automatically operate (manage) the components and applications of a

machine environment. IT management in data centers, in contrast, still largely depends on human labor today.

It is a valid question to ask why this is still the case and what could potentially be learned and adopted from operating systems to help construct systems that can perform similar functions in the context of a data center – functions such as resource management, configuration, deployment, run-time control, coordination, performance management, failure handling and recovery.

And yet, there are substantial differences to consider with regard to size, scale, lifetime and lifecycle, diversity, granularity and complexity of hardware and software components in data centers and in machines that need to be taken into account. These differences are of such a substantial nature that management has emerged in fundamentally different ways. It is automated and performed by a system in the case of operating systems and it is mostly manual in the case of data centers. No system has emerged that comprehensively, transparently and automatically could manage the IT infrastructure and applications in a data center like an operating system manages these in a computing machine.

The goal of this thesis is to develop the architecture of a system that automates the management of IT infrastructure in a data center. The approach for developing this architecture is based on connecting concepts from operating systems with concepts from IT management. Both disciplines have been considered largely independently in the past. The idea of connecting them and consolidating the result in form of an *Architecture for the Automated Management of Data Center IT Infrastructure* is new and the contribution of this thesis. It has not been published before and is the sole and original contribution of the author.

The thesis is founded in research conducted by the author at HP Labs between 2003 and 2008 in the field of data center automation. The motivation for this thesis originated from the desire to consolidate the insights obtained during this research and summarize them systematically in form of an architectural framework.

1.1 Operating Systems

An operating system is a software system that manages the defined and coordinated execution of a set of applications on a set of shared hardware components of a computing machine. An operating system is a system that operates: 1.) the system of applications executing in the environment; 2.) the system of hardware components on which applications execute and 3.) the operating system itself, which is executing in the same environment.

For operating systems, the computing elements are the components of the machine provided for local processing, storage and communication such as: CPUs, caches, memory, storage, controller, busses, interconnects and external interfaces. In order to execute applications in this environment, the operating system must turn machine components into *resources* that can drive application processes. This occurs by properly configuring components and controlling them during operation through their control interfaces – tasks that are performed by the operating system transparently and autonomously without requiring attention from applications or users.

The need for utilizing computational components of a machine economically led to the desire to perform multiple computations simultaneously and share the machine

components among applications. The principles of *sharing* and *concurrent use* were fundamental for operating systems. They required additional coordination, which soon became automated in the operating system. Over the course of time, comprehensive coordination mechanisms and abstractions were developed in the field of operating systems. Time sharing was an early technique to multiplex multiple applications on a set of shared computational components. Virtualization was a more advanced multiplexing technique by virtually creating more component instances than actually existed in the environment and providing applications with the impression of their exclusive use.

Dynamic provisioning of resources was another core principle to supply only those quantities of resources to application processes that were actually used. Rather than assigning static portions of resources to applications – configured for maximum use – these assignments became flexible such that the operating system could observe actual workloads and dynamically adjust resource allocations during run-time accordingly. This further increased the resource utilization of a machine. It required that applications and data became decoupled from the locations in computing components and created the need to develop *resource abstractions* and abstraction layers allowing applications to achieve this independence. The mappings between the resource abstractions – visible in the application layer – to the actual (flexible, adjustable) locations in machine components had to be managed, which became another task automated by the operating system.

Automation implies decision making. Operating systems employ *policies* to guide automated decisions such as for resolving resource contention conflicts. Policies are another important principle used in operating systems to achieve automation.

To summarize, fundamental principles of operating systems with regard to automated management are:

1. the transformation of hardware components into *resources* by properly configuring them and assigning them to applications driving their processes,
2. the *sharing* and *concurrent use* of resources in a coordinated and transparent manner,
3. the *dynamic provisioning* of resources depending on actual use,
4. the creation of *resource abstractions* and the decoupling of applications and data from locations in underlying physical components with the transparent and automated management of the construction and mapping relationships by the operating system, and
5. the use of *policy* to guide automated decisions.

These principles have been established in operating systems for a long time. In the data center, in contrast, most of these principles are just emerging. Dynamic provisioning is a recent example motivated by the same reason as it was for computing machines: to more effectively utilize the resources of the data center by sharing them and dynamically adjusting resource supply according to actual demand rather than the static provisioning for the maximum (peak) demand case. Introducing virtualization into data centers is another example. It allows decoupling applications from physical components in order to manage them more flexibly and in a more compact manner.

The introduction of dynamic techniques of provisioning and virtualization into data centers creates problems. It breaks fundamental assumptions in IT management and in IT

management systems, such as the definition of the existence and the identification of components and resources. Today's IT management systems mainly recognize physical components as existing using physical properties for their identification such as physical network interface addresses. Those assumptions fail in virtualized environments.

Recognizing virtual resources, such as virtual machines or disks, that are visible in the environment only at times when they are active, which can change locations, and which do not possess physical prosperities for identification, cause significant difficulties to management systems. Management of virtual resources is thus deeply fragmented and not transparent. For example, virtual machines can be created by a virtual machine management system. But placement decisions onto physical machines are made outside this system. Other resources that are needed for virtual machines, such as networks and disks, need to be created and configured in other management systems. Overall, it is left to people tying all the fragmented information together for managing the entirety of physical and virtual resources in the environment. [Gra03] describes implications virtualization has on IT management systems.

Operating systems faced similar problems when resource sharing and dynamic provisioning were introduced in computing machines in the 1960's. It led to a new understanding of the concept of resources beyond physical machine components and to the introduction of resource abstractions and the explicit recognition of resource construction and transformation processes performed by the operating system. Mapping and construction relationships among resources were explicitly represented and managed in the information model of the operating system allowing the operating system to automate the construction processes and making them transparent to the higher layers.

These insights from operating systems have not yet been adopted in IT infrastructure management in data centers. But the trend towards more dynamic resource management and automation in data centers can mark the inflection point where principles from operating systems can influence future data center management systems and practices.

1.2 IT Infrastructure Management in the Data Center

As stated earlier, the overall goals of operating systems and IT infrastructure management in a data center are essentially similar: to execute applications and deliver computational services in a controlled and coordinated manner. Like in operating systems, portions of data center components must be selected, configured and transformed into resources that can drive applications. In a data center, these activities are performed by organizations of specialists who are concerned with decision-making and execution of IT management tasks. Management systems are directed towards supporting people, they provide little support for autonomous decision making and management task automation.

The management of a data center is a complex and thus fragmented task. Higher-ordered tasks, such as resource allocation and scheduling, are performed as planning tasks by one group of specialists. Another group is concerned with configuration and deployment, while yet another is in charge of operational management. Different specializations have emerged for different domains. Servers, networks and storage are largely managed separately by different groups of specialists today.

It is a valid question to ask why proven principles from operating systems have not yet been adopted to the infrastructure management in data centers? Reasons are manifold.

1.) One major reason is that a machine usually is acquired as one unit and its hardware configuration is rarely changing over its lifetime. This provides the operating system with a stable set of components that are known up front. It is easy to establish an inventory of all components upon which the operating system can operate. The operating system maintains a consistent information model of the entire environment at all times. The automated algorithms an operating system performs fully rely on this consistent and comprehensive information model, which is represented as a set of centrally organized in-memory and persistent data structures as part of the operating system.

2.) Since data centers exist for longer periods of time, they face a continuously changing inventory, mainly driven by overlapping equipment and application refreshment cycles. This dynamism makes it difficult to maintain a similarly consistent information model for a data center. Due to the fragmentation of management tasks and systems, information is scattered and duplicated. Management systems typically coexist in isolation in a data center and are unaware of one another. For example, a storage management system has little insight into networks, which are managed by a network management system. It is hard to consolidate the management information across these systems and achieve a consistent state. The lack of standardization furthermore impedes integration of management information and systems.

3.) Today's IT management systems are constraint to existing physical inventory relying on permanent physical presence for discovery and detection of physical properties for identification. These assumptions fail when resource abstractions (such as virtualized resources) are introduced.

4.) IT management systems are proprietary, fragmented and incompatible. The separation between component vendors and management system providers has not occurred like it has for computing machines where (hardware) component vendors and system software providers have separated and specialized creating the need for common abstractions and interfaces allowing systems and software from different vendors to be combined.

5.) Common abstractions and interfaces enabling the integration of management systems for hardware and software components have not emerged. The Hardware Abstraction Layer (HAL) in the operating system is an example of such an abstraction layer providing a common interface to underlying hardware components isolating their diversity from the device driver code. The HAL enables the development of reusable algorithms performing higher-ordered management tasks such as scheduling and resource management. Only few comparable abstractions have emerged in the context of data center management.

6.) Data centers with their overlapping planning, design, deployment, operation, and retirement cycles – occurring at all layers simultaneously – require more comprehensive information models than currently are available. New concepts must be incorporated such as future states (plans, designs), the transitions of those states into reality and their subsequent operational management. It is not sufficient to only represent the current state of physically present components. No automated conclusion can be made when only the currently observed state of a component is known, but not what the expected state of that component is. Furthermore, complex resource constructions with mapping and construction relationships must be represented. Developing these concepts means that management information models must be significantly extended with new expressions.

To summarize, the lack of a comprehensive and consistent information model in IT management is a key obstacle to automation. Establishing this information model is the prerequisite for constructing algorithms that are performing management tasks rather than people. New management abstractions need to be introduced to reflect broader notions of resources including constructions of resources. Future states must be represented, such as plans and designs, enabling the automation of construction and transition processes. Addressing these issues is fundamental for the development of the architecture of a data center IT infrastructure management system.

1.3 Different Approaches to Innovation

While operating systems have focused on developing algorithms and the construction of software systems automating the operation of applications in computing machines, advances in IT management have focused on organizing the work of people in IT management. Comprehensive overviews of IT management are provided in the literature [Heg94], [Heg95], [Blan99], [Heg00], [Cass00] and [Hol00].

A large body of guidelines and frameworks has evolved for IT management summarizing the experiences gathered over decades of IT management in form of guidelines and best practices. The Information Technology Infrastructure Library (ITIL) [ITIL] is a prominent example of such a framework. It is directed towards IT experts helping them to organize their work for a broad spectrum of activities in IT management, ranging from planning IT strategy; designing IT services; mapping them into IT systems; transitioning them into data centers followed by subsequent operational management and continuous improvement. ITIL employs the abstraction of a *process* as a unit of work that needs to be performed for a management task. Policies (as directives for decision making) are provided for human decision makers who follow them in their daily work.

Other examples of management frameworks are the Control Objectives for Information and related Technology (COBIT, [COB]), the enhanced Telecom Operations Map (eTOM, [eTOM]) and ISO/IEC 20000 [ISO2K]. Those frameworks can also be used to structure the complex task domains of IT management. Some of these frameworks have originated from the telecommunications industry. Attempts have been made to consolidate and integrate the different frameworks. This topic is subject of continued research in IT management [Dre02], [Nai04], [Scha07].

More recently, broader contexts are considered for innovation in IT management, such as considering business aspects of IT [Sauv09]; the mapping of business requirements into IT requirements [Bart04a], [Sauv06], [Gra08]; quantifying risk in IT [Sauv07] and the external sourcing of IT services [Joch05]. Energy consumption and sustainability are further domains in data centers where interest and innovation are emerging [Schie01], [Sne02], [Gma10].

1.4 Problem Statement

Most automation efforts in data centers fail or fall short of expectations. The underlying fundamental problem is the lack of a systematic approach to data center automation and the lack of a comprehensive architecture that is founded in established principles and provides a set of abstractions based on which management tasks can be automated. This thesis addresses this problem.

1.5 Hypothesis

The hypothesis of this thesis is to demonstrate the systematic development of an *Architecture for the Automated Management of Data Center IT Infrastructure*. The systematic approach is rooted in adopting concepts from operating systems and adapting them to the domain of IT infrastructure management in a data center. The architecture builds on a set of abstractions based on which a broad set of management tasks in a data center can be automated.

1.6 Questions

This thesis seeks answers to the following questions:

1. What are the fundamental problems of data center IT infrastructure automation?
2. Can a comprehensive set of requirements be formulated for the automated management of data center IT infrastructure?
3. How can data center infrastructure automation systematically be achieved?
4. Which abstractions need to be developed upon which the architecture for the automated management of data center IT infrastructure can be built?
5. Which concepts and techniques from operating systems can be adopted for data center IT infrastructure automation?
6. Which concepts from IT management must be accommodated, which are not present in operating systems?
7. Which new abstractions and concepts must be developed that can neither be derived from operating systems nor from IT management?
8. How can the information model for these abstractions be defined?

This thesis will use the term **Data Center Infrastructure Operating System (DCI-OS)** to refer to a software system that provides capabilities for the automated management of data center infrastructure. The DCI-OS focuses on data center infrastructure management recognizing the fact that there are other domains of IT management, which cannot be automated and which are not addressed.

1.7 Outline

This thesis first discusses a selection of fundamental concepts from the disciplines of operating systems and IT management in Chapter 2 and 3, respectively. Key concepts are identified from both disciplines. The next step is the formulation of a set of requirements for a DCI-OS based on the discussion of an early HP data center automation solution, which had failed, in Chapter 4.

Based on these inputs, the DCI-OS architecture is developed and its abstractions are presented in Chapter 5.

The following chapter presents realizations that were developed between 2003 and 2008 for purposes of validating the research. Experiences obtained from pilot implementations with HP product groups are documented in Chapter 6.

Since the architecture is structured in layers, the content of the three main layers is then presented in detail in Chapters 7, 8 and 9, respectively. The discussion in these chapters

summarizes research and results from prior publications and relates them back to the architectural framework. Chapter 10 summarizes the work and concludes the thesis.

The outline of this thesis is summarized:

1. Introduction.
2. Concepts from Operating Systems.
3. Concepts from IT Management.
4. Combination and Requirement Analysis.
5. Architecture.
6. Realization.
7. Layer 1: Data Center Planning and Design.
8. Layer 2: Infrastructure Services Layer.
9. Layer 3: DCI-OS Layer.
10. Summary and Conclusions.

Chapter 2

Concepts from Operating Systems

2.1 Definitions

This thesis uses a broad definition of the term operating system:

*An **operating system** is a **software system** which operates other systems.*

In a machine environment, operated systems include:

- a system of **applications**, which is operating on
- a system of **hardware components**, and
- the operating system itself is a system which also operates itself.

This general definition can be applied to a machine environment, as it traditionally has been, but it can also be generalized and applied to the broader scope of a data center environment where systems of applications need to be operated on systems of hardware components found in the data center.

An operating system conceptually resides as a **layer** between the layers of programmable machine components and the layer of application software. An operating system provides the **application execution environment** in which applications can execute. It furthermore transforms machine components into **resources** by properly configuring and controlling them. It in turn transforms resources into resource **abstractions** providing a more comforting execution environment to applications and also isolating applications from diversity found in resource properties. The entirety of abstractions is also referred to as **abstract machine** that is produced by an operating system as a software machine. An abstract machine differs from a virtual machine although both are realized through software. An abstract machine produces new abstractions and new capabilities for applications which are different compared to the capabilities found in the physical machine environment. In contrast, a virtual machine replicates the full behavior of the physical machine with the same properties as found in the physical machine environment. Virtual machines allow the same applications to execute that were written for the physical machine environment.

An operating system isolates applications, controls their execution, prevents interferences between them and resolves resource contention conflicts while managing the resources of the environment. It also manages itself fully automatically and autonomously.

A **Data Center Infrastructure Operating System (DCI-OS)** is defined as a software system which operates other systems in a data center such as:

- the system of applications in the data center executing on
- the system of active data center components (as active components are understood components which can participate in computing tasks, such as server, storage and network components).

Data center components are transformed into a system of resources that can be supplied into the execution environment of an application system allowing it to execute.

Today, the operational management tasks a DCI-OS would perform are carried out by teams of people who are involved in the planning, the design, the creation, operation and the management of the components and the applications in a data center.

The hypothesis of this thesis is that it is possible to define the architecture for a technical system that can automatically perform those operational management tasks for a data center environment and hence would qualify as a DCI-OS.

2.2 The Evolution of Operating System Concepts

Purpose of the discussion in this section is to identify essential theoretical and practical concepts that have emerged in operating systems over the past 50 years and that have led to the class of technical systems which we know as operating systems today. A broad overview of the progress of this field is provided in [Han01]. The fact that operating systems have become an established field is also demonstrated by the broad availability of text books such as [Kal90], [Wett93], [Tan01], [Silb02], [Deit03] and [Stal04].

The discussion here addresses a boarder scope of concepts than usually is associated with operating systems, particularly in the later part of the section. It includes application-level services and distributed execution environments, which are not part of traditional operating systems. A discussion of these extended architectures and their influence on application architectures can be found in [Gra97].

A variety of categories of operating systems have emerged over time. In the early years of electronic computing (1950's-1960's), computing machines required permanent attention and were fully operated by people. The profession of an **operator** emerged. The role of an operator was different from the role of a **user** because it mainly dealt with operational tasks such as the allocation of memory to a program, loading the application code, controlling the application's execution, providing input and directing output to the desired medium. All these tasks were performed manually by operators. These tasks of basic resource management and application control are typical tasks that are still performed manually in data centers today.

Beginning in the 1960's, operating systems emerged from **libraries and tools** provided by machine vendors that supported the development (libraries) and management (tools) of applications on their machines. Libraries primarily supported programming on machines. Tools were used to support people in their operational tasks. This still is largely the situation how data centers are managed and operated today.

An important mile stone were the early monolithic operating systems for main frame machines which truly could be called operating systems since they operated themselves and carried out basic machine management tasks. Operating systems emerged as result of

integrating reusable tools and libraries into a coherent system that provided a common set of functionality that was necessary to operate a machine.

Example of those early operating systems were IBM's OS/360 (and later series 370, 380 and 390); DEC's TOPS and VMS; HP's MPE for HP3000. Since hardware was expensive, **sharing** was an important goal of operation. Essential concepts were introduced such as **time-sharing** and **multi-tasking.** Time-sharing and multi-tasking required a concept of **isolation** between coexisting applications on a machine and **controlling resource use** (**quotas, partitions**).

Virtual machines evolved to virtually replicate the entire hardware of the machine itself within which multiple instances of operating systems could coexist.

Persistent data management was a core capability operating systems developed from the very beginning, mainly relying on block-oriented external storage (disks, tapes, punch cards or tape). Blocks were organized into **files** and various models were created to organize files into file libraries or **file systems**.

Another important concept emerging early was **resource transformation**. **Resource generation** allowed producing higher-ordered resource abstractions from more basic resources such as a file system abstraction created based on block devices. **Resource virtualization** allowed the multiplication of resources for use by multiple applications or expanding resource capacity by mapping one resource to another. An example is virtual memory which expanded the physical memory of a machine by swapping ranges of memory between main memory and a block device. Resource transformation requires active processing to take place. The operating system performs this processing itself or initiates it in underlying active components by properly instructing and configuring them.

Theoretical foundations in operating systems were also mainly laid in the early 1970's by Dijkstra's work on **parallel processes** and **process control** (THE System, inter-process synchronization and communication mechanisms such as semaphores) [Dij68], [Dij71] and C.A.R. Hoare's work on parallel programming (CSP, Communicating Sequential Processes) [Hoa78]. A large body of research exists in areas of resolving **resource contention** and algorithms for resource **scheduling**. Denning investigated the phenomenon of locality of memory access in von-Neumann machines developing it into the theory of **working sets** [Den68] which enabled virtual memory. Haberman introduced **architecture** to operating systems in form of layers of hierarchical functions. Specifications of functions allowed defining **interfaces between layers** and separating concerns addressed in layers [Hab76].

There has always been a close relationship between operating systems and computer architecture. For example, one aspect that made operating system software special from other software was that it "managed itself". It had to manage the same resources it itself was running on, the same processor, the same memory and the other resources it used and shared with the applications. In order to enable this, computer architecture developed an important synchronization mechanism between hardware components and the processor in form of **interrupt signals** received by the processor from the **interconnect fabric** in the machine which consisted of wire connections between machine components and the central processor. Processors were designed such that they were able to receive those signals at any time and respond to them by a special cycle interrupting the currently executing application and handing execution over to an interrupt handling routine in the

operating system. The concept of **preemptive transfer or control** back to the operating system in response to a condition that had been signaled was critical for the operating system to gain control back when application programs were executing on the same processor it itself was executing on. This principle has become common across all generations of processor architectures and is still present today.

After vendors had established successful lines of main frame machines and main frame operating systems, Ken Thompson planted the seed of a new class of operating systems at Bell Labs by porting a smaller set of the Multics experimental operating system to a PDP11 [Cor65]. Multics had been particularly focused on a new security architecture for operating systems. It primarily introduced **security mechanisms** of **protection** and **access control** in a **multi-user** environment, which Thompson (in a reduced form) carried over to what would become the Unix family of operating systems. It led to the separation of a protected kernel part and the isolation of application parts that were also protected among each other. By the clear separation between kernel and applications, an **operating system interface** could be defined between them.

It was at that time that a strict physical separation between the operating system and the application environment was introduced. The operating system provided a common application execution environment for all applications running on a machine. The development of applications was fully decoupled from the operational tasks of user, resource and process management performed by the operating system.

User management introduces the concept of a user into the system which is needed to establish mechanisms for protection and access control, resource use accounting and resource use control. **User identification** forms the basis of user management typically using some form of user **credentials** such as passwords.

Operations for **process and resource management** were defined around basic operating system abstractions such as file systems, **processes, inter-process communication** (IPC) and **file-based IO** mechanisms, which still describe the essential abstractions used in operating systems today. The Unix operating system developed ways to integrate those capabilities into the existing file system abstraction.

Another trend in the 1970's and 1980's was the upcoming of powerful parallel machines with specialized architectures that required specialized operating systems. Examples were COS and CTSS for Cray for numerical processing or TOS for Tandem (now HP NonStop) for parallel transactional processing. Scale and **scalable processing capacity** had been essential drivers for those developments.

Connecting computers through **networks** had been another major trend in the 1980's. Operating systems incorporated networking support into their core capabilities supporting essentially all networking technologies that had been emerging over time (the Internet protocol suite, but also vendor-proprietary networks that gained wider use such as DECnet, IBM's Token Ring and SNA, Novell, etc.). Accessing machines by users over a network was an initial advantage which was enhanced by adding support for accessing remote machine resources and enhancing inter-process communication across machine boundaries. It enabled a new class of **networked applications**. The most widely used and adopted early networked applications were file transfer and e-mail.

The 1980's were also characterized by the miniaturization of computers and the upcoming of the personal computer. Apple's MacIntosh had the first operating system

that introduced a more user friendly graphical **user interface** and new human interaction devices such as the mouse building on research Xerox PARC had done on SmallTalk earlier. Microsoft's coup with IBM to develop a simple operating system for the IBM PC laid the foundation for the Windows operating system family which is dominating the market for personal computer systems to this day.

Due to the growing diversity of operating systems and increasing effort to port applications from one system to another, standardization became important. In earlier years when operating systems had been proprietary to the machine vendor, vendor-specific unification of operating systems had been the dominating trend. IBM, for instance, has established a reliant line of compatible operating systems over generations of technologies from its main frame systems to mini computers that are still capable of executing applications that are decades old. Application software had also become a significant investment for customers that needed to be protected. **Portability** and **compatibility** of applications to new generations of systems had become essential.

Unix was the first operating system that was adopted across machine vendor boundaries. Customers for the first time could port applications from one system vendor to another and overcome the application lock-in to a particular system vendor. Vendors first aimed to gain control over Unix by creating proprietary derivates (AIX, HP-UX, Ultrix, SunOS/Solaris) and fiercely competing over proprietary capabilities, which became also known as the era of the "Unix wars".

However, the need to keep applications on Unix portable, manageable and interoperable led to the demand for **standardization**. ANSI became the body to standardize the C programming language and libraries. IEEE became the organization for standardizing the Portable Operating System Interface (POSIX). Standardization led to commodization and consolidation of operating system vendors, a trend that continues to this day.

The 1990's were characterized by the spread of networks and the Internet. It is thus not surprising that this decade was also the decade of distributed operating systems. Networks developed at four major scales: intra-machine (to build machines with large numbers of processors), local between machines (LAN), at campus or metropolitan (MAN) and wide area scales (WAN). The networks and technologies connecting components at the four scales were substantially different.

The need for parallel computing drove the emergence of NUMA (non-uniform memory access) architectures that allowed better scaling of CPUs than SMP (symmetric multiprocessing) architectures. Fast intra-machine interconnects were needed. Accessing a large number of CPUs to shared intra-machine memory efficiently along with work on parallel algorithms for those architectures defined a large body of academic and applied research in the 1990s. Burroughs, Convex Computer, SGI, Sequent and Data General built NUMA machines which, particularly later, used specialized Unix operating systems with built-in NUMA support. Access from a larger number of processors to memory with different access characteristics for performing large parallel applications was the main contribution of this class of systems. An interesting alternative at this time was the Transputer architecture from Bristol-based INMOS using processors with local on-chip memory equipped with 4 programmable communication paths to other processors allowing multi-dimensional interconnect topologies between processors such as hypercubes rather than presenting an abstraction of one large shared memory resource.

In the late 1990's, distributed operating systems shifted focus from intra-machine parallel computing to inter-machine computing accelerated by the rapid scaling of the Internet. Representatives of academic operating system research of that era have been Amoeba [Tan91], BirliX [Härt90], Condor [Cond96], Eden [Laz81], Locus [Pop85], NOW [NOW95], V [Cher84] and Accent [Rash86]. Text books covered the field of distributed operating systems [Gosc91], [Tan94]. Essential concepts from those operating systems were the ability to access resources on remote computers as well as the ability to **migrate process execution** to other machines over the network (e.g. in BirliX, [Härt90]).

In the domain of parallel numeric computing, high cost of NUMA and SMP systems, which both primarily still meant processing in one machine, had led to the emergence of cheaper alternatives in form of clusters of standard computers connected over high-speed interconnects. Clusters still were operated in one location such as in one data center.

Grids [Fost98] then introduced WAN connectivity between clusters and established a global computing environment for parallel applications. A number of large Grids exist today. Grid computing has contributed tools and systems for **automated resource management** in clusters. Examples are schedulers such as Platform's LSF [LSF] or the Maui scheduler [Maui] for the popular PBS batch processing system [PBS], which automated the allocation of batch jobs to available nodes in a cluster. Simple automated tools for **application deployment** have emerged as well as have tools for remote administration of large numbers of nodes such as Cfengine [Burg93]. More advanced academic work in Grids has recently focused on market-based mechanisms, which may form the basis of a new category of cloud or utility computing environments.

The most important concept from Grids is the global **federation of resource environments** in which applications can be executed transparently from the actual location. Programming libraries such as PVM [PVM] and MPI [MPI] emerged and are still popular today to develop parallel, message-based distributed applications under the premise of **location transparency**. While Grids have pushed the overall concept of federated computing far, it still mainly is focused on connecting distributed clusters of similar machines for the class of batch-oriented processing. **Enterprise Grids** [Rob04], [Gra04a] aimed at expanding the focus of scientific Grids on numerical processing to the broad range of enterprise applications predominantly running in enterprise data centers. The Enterprise Grid Alliance (EGA) [EGA] was formed.

This "new" [Fost03] Grid has replaced the proprietary middleware used in earlier Grids with web services standards defined in the Web Services Resource Framework (WSRF) [WS-RF] and Web Services Distributed Management (WSDM) [WS-DM]. The new grid also introduced broader ideas of a general Open Grid Services Architecture [OGSA] that could be expanded into other domains than scientific computing. One of these domains is geographically disperse enterprise data centers providing resource pools for enterprise applications and services. Although a number of standards have been developed, the overall idea of the new Grid did not materialize in the enterprise.

Back in the second half of the 1990's, concerns about the complexity began to arise referring to the growing sizes and complexity of operating systems, particularly with respect to stability and security of those "big" kernels in networked environments leading to renewed discussions about operating system architectures. Reducing and shrinking operating system kernels to a minimal set of essential functionality became the focus of

operating system research in the second have of the 1990's. Dynamic adaptability was another field that was explored. It meant the ability to load and unload code into the operating system only when needed. The miniaturization of operating system kernels led to micro- (Mach [Acc86], SPIN [Ber95]), pico- (Panda [Ass93], KeyKOS [Bom92], Choices [Camp93]) and nano-kernels (L4 [Lie95], ExoKernel [Kaa95]), which reduced the operating system functionality to a minimum of essential mechanisms such as context switches, basic memory management and inter-process communication.

Software components such as device-drivers, were placed outside the inner kernel executing themselves as **application-level services**. This concept of providing operational functionality not from within the operating system kernel, but at the same level as the applications would turn into an important pattern of how large networked environments are composed and operated today. The distinction between **mechanisms** and **policies** was an important insight from this research.

A side line of the trend of miniaturization led to specialized operating systems for embedded and real-time applications. Those operating systems introduced scheduling especially tailored to meet hard or soft (statistical) reaction time guarantees, which was called **real-time scheduling**.

In the industry, however, computing over the network in the late 1990's was not approached from the operating system perspective. It was instead approached in form of **application stacks** that allowed applications to access other applications remotely. Layers in those stacks aimed at decoupling functionality and hiding the exposure of changes in one layer from other layers (as long as interfaces were not affected). **Layered architectures** have proven successful in networked environments and form the architectural pattern of distributed environments to this day.

Distributed programming environments were developed such as the Distributed Computing Environment (DCE) [DCE] defined by a consortium of Apollo, DEC (both later HP), IBM and Sun in the Open Software Foundation (OSF) [OSF] which later became The Open Group. DCE was not just a set of libraries and tools that supported remote procedure calls among applications. It also had a substantial run-time (or operating system) capability. It offered a Distributed File System (DFS) [DSF] with Kerberos-based authentication and ACL-based access control as well as a number of application-level services such as time, directory and other run-time services needed to support a distributed operating environment. Microsoft developed with DCOM [DCOM] a similar platform for its distributed applications. With the emergence of object-oriented programming, the Object Management Group (OMG) [OMG] introduced object-oriented programming into distributed application environments with the Common Object Request Broker Architecture (CORBA) [CORBA], particularly for applications written in popular C++. With the emerging popularity of Java, Sun rebuilt a stack of distributed programming capabilities for web and web services applications. This stack is known today as the Java Platform, Enterprise Edition or Java EE or J2EE stack [J2EE].

DCE, DCOM, CORBA and later J2EE had a further diminishing effect on the importance of local operating systems in a distributed environment. The role of what is commonly understood as operating system has remained limited to managing the local resources of a machine. The traditional class of operating systems with its local kernel and local applications has little significance for the operation of larger distributed environments.

Rather, **distributed services** form the architectural foundation of those environments, and consequently, a **distributed operating environment** is also the architectural pattern of a data center environment.

To summarize, operating or managing a larger distributed environment such as a data center leads to another dimension of systems. This new class of operating systems is broader than what is considered as a traditional operating system today, which manages resources and application processes of a local machine. Functionality to manage or operate a larger distributed environment is provided in form of distributed services, which are applications themselves from a local operating system point of view.

Management services exist today in simple forms such as a simple resource manager in a Grid cluster such as GRAM [GRAM] or timing, naming or registry services in distributed environments such as CORBA. However, most management tasks are left to operators.

2.3 Categorization of Operating System Concepts

The previous section highlighted a number of concepts in bold font. These concepts are considered operating system concepts that are now categorized in Figure 1. The figure shows an architectural diagram often used in operating systems with three layers:

- the **Application Layer**,
- the **Operating System Layer** and
- the **Hardware Component Layer**.

System architectures using layers have proven to be powerful. They are thus suitable to structure complex hardware and software systems in general. Layers separate domains which build one upon another and establish well-defined interfaces and abstractions within and between them. Interfaces mediate direct interactions between components in different layers. Abstractions refer to the computing entities which exist in each layer, their properties and relationships. Usually, an underlying layer produces the abstractions or computing elements of the higher-ordered layer. For instance, the abstraction of an application process, which exists in the Application Layer, is produced in the underlying Operating System Layer as result of computations in this layer, such as processor scheduling and address space creation by programming the memory management unit of the processor.

The pattern of layers supports separation of concern, isolation, encapsulation, modularization and interaction, which are all desirable properties of complex software systems. It also allows decoupling and specialization. It has essentially created an industry of vendors in computing systems today which have specialized in making hardware components, operating systems and application systems, which at the end fit together into one complex system. This specialization has not occurred in IT management mainly due to the fact that the concept of layers has not sufficiently been adopted.

Figure 1 shows the three layers of an operating system with essential components and abstractions within them.

Application Layer

Application Layer		**Application Environment**: application processes and application data; multi-user, multi-tasking environment;
User	App-Intf	
Operator (Admin)	Admin-Intf	**Application Execution Environment** (abstract machine) with abstractions: user, process, file system, inter-process-communication, protection, resource isolation (quota, partition), virtual machines and access control;

Operating System Layer

Operating System Layer	**Operating System Interface**		
	User Management: user identification, role, credentials;	**Process Management**: process control (lifecycle), isolation, inter-process-communication;	**Persistent Data Management:** file system, special file-based IO and IPC;
	Resource Management: resource creation and transformation (generation, virtualization), sharing, scheduling, resource contention policy, accounting and controlling resource use (isolation, partitions, quota), protection (access, integrity);		
	Hardware-Abstraction Layer (HAL) with generalized component interfaces;		
	Drivers apply control instructions and configuration onto components and report their states; signals and interrupt handling;		

Hardware Component Layer

Hardware Component Layer	**Component Interfaces**: ports, registers, memory-mapped access;
	Components (Resources, Devices, Interface cards): processors, memory, MMU, controllers, network and other periphery interface cards;
	Interconnect Fabric: busses, connection links, DMA, interrupts, signals;

Figure 1: Categorization of Operating System concepts.

2.3.1 The Application Layer

The application layer contains concepts such as a user role, the application user interface, application processes and application data. General concepts are multi-user and multi-tasking environment. A specific role of an operator (or administrator or super user) has privileges to carry out administrative tasks on the system and in the application execution environment. Further concepts include the following.

- The *Application Environment* contains the applications, their processes and data. Multiple applications co-exist and perform simultaneously sharing components of the machine environment on which they execute.

- The *Application Execution Environment* provides the resources presented to applications enabling them to execute. Those resources are produced by the underlying operating system layer such as file systems, processes, inter-process-communication, partitions, virtual machines, isolation and protection mechanisms.

Both, the application environment as well as the application execution environment provide interfaces to administrative roles for setting policy. User roles interact with applications through user interfaces. Operator or administrator roles interact with the application execution environment through administrative interfaces.

2.3.2 The Operating System Layer

The *Operating System Interface* decouples the application layer from the operating system layer. One specific property of a traditional operating system is that it, as a software system itself, executes on the same machine as the applications it operates. Special protection of the operating system software is enacted by switching the processor into a privileged mode when executing operating system code. Special machine instructions allow system calls to switch between the privileged kernel mode and the less privileged user mode under which applications execute as part of the system interface.

Within the operating system layer, there are three main functional blocks providing the essential operating system functionality. These functional building blocks comprise:

- *User Management* – implements functions supporting the concepts of user and role. These functions include identifying a user, assigning roles to users and issuing and managing credentials. Purposes of user identification are to control access to operations and resources as well as to account and control resource use.
- *Process Management* – implements functions supporting the concepts of processes executing applications. Process control includes basic lifecycle operations for processes such as create, start, activate, pause, suspend, resume, interrupt (signal), stop and terminate. Processes are isolated by separating their address spaces. Inter-process-synchronization and communication required special primitives within a machine and across a network.
- *Persistent Data Management* – implements functions supporting the concepts of persistent data. Most modern operating systems have adopted the abstractions of storing data persistently in an abstraction called files, which are organized as hierarchical file systems. In addition, input and output operations as well as primitives for inter-process communication which can be mediated through file-based IO abstractions are included here using special file types.

Since all functionality of an operating system relies on resources, all three functional blocks rely on a layer of:

- *Resource Management* – implements functions needed for creating and transforming resources and for supplying them into application execution environment allowing applications to execute. Resource Management includes a number of tasks:
- *Resource Creation*. A resource itself is an abstraction which is created by an operating system. Resources are created by a transformation and a mapping onto machine components which partially or wholly occupy them. At the component-

level, resource creation means to apply specific configurations to the component and load its state onto the component, if this is required. Application processes can only proceed when all their needed resources are mapped onto actual machine components (all-or nothing policy).

- *Resource Transformation*. Resources cannot only be created by mapping them directly onto machine components, such as a memory resource of an application process mapped onto a specific address region in RAM. Resources can also be transformed into new resources with new and different qualities. Two kinds of transformations are distinguished:
  - *Resource Generation* is the process of generating resources with new qualities and new properties based on more elementary resources. An example are file and file system resources, which are created based on elementary block resources mapped onto a disk or another persistent storage component of the machine environment. A file resource is generated by managing additional mapping information of a logically constituent data set (the file) onto a sequence of block resources.
  - *Resource Virtualization* is the process of multiplying underlying resources with the same properties. Purpose of virtualization is the exclusive assignment of virtualized resources to processes when resources are mapped onto shared underlying components. The mapping and sharing complexity can be hidden using virtual resources. Another reason is the decoupling effect virtualization has for the applications themselves avoiding potential interference among processes in a shared environment (e.g. configuration overlaps on shared machines can be avoided by placing overlapping applications into separate virtual machines).
- *Resource Sharing*. Multiple resources are typically created on a set of shared machine components. Two patterns can occur for sharing. One is sharing in space, when two resources are mapped onto different regions of an underlying machine component such as into different address regions in memory. The other is sharing in time, which is the typical time-sharing or multiplex pattern. Here, two or more resources cannot co-exist on the same machine component. Hence, their state must be switched such that only one resource actually resides on the component and the others only exist in form of their state representations in another machine component which can store state. CPUs are typical components that are shared as processor resources.
- *Scheduling* is the process of determining the assignment schedule over time of resources to underlying machine components, which does not only apply to processor resources which must be mapped onto CPU. Scheduling is a concept that generally applies to coordinate the mapping of time-shared resources onto machine components.
- *Resource Contention Policy*. Since most machine components are shared by resources, their mapping must be scheduled. Resource contention is the effect when application processes compete for resources and their mapping onto shared machine components. It is often the case that more application processes could proceed if all their needed resources could be mapped onto machine

components. A resource contention conflict occurs when there are less mappings available than resources needed by application processes. In this case a decision must be taken, which applications processes will be granted their resources and which ones have to wait. The resource contention policy is often applied to resources of the same type, e.g. processor resources or memory resources. Processor resources are often managed with a policy of fairness among all application processes. Other policies can honor prearranged priorities or meeting defined completion times or throughputs (processing units/time).

- *Accounting and Controlling Resource Use*. It is often required that resource use is accounted and controlled on a per-user basis avoiding users monopolize resources by launching large numbers of processes and allocating resources such as file systems by isolating processes from one another using mechanisms such as resource partitions. Operating systems provide for this purpose means to account resource use and also set policy limiting resource use by enacting quotas (such as maximum quantities of resources allowed to use by users).
- *Protection.* Operating systems also provide means to protect resources from unauthorized access as part of resource management. Access control relies on the identification of users which are seen as subjects accessing resources. The other dimension of protection is ensuring resource integrity which means preventing unauthorized alteration of resource state.

Resource management operates over elementary resources, which result from the direct mapping onto machine components such as processors, memory with address spaces and physical as well as byte- or block-oriented access to I/O devices (e.g. disks). Higher resource abstractions are generated from sets of those elementary resources.

Resource management of an operating system interacts with the physical components of the machine environment through their control interfaced. In order to support portability of the core operating system software itself, an internal hardware abstraction layer is introduced in modern operating systems. This sub-layer of an operating system isolates the higher building blocks from detail and diversity found in hardware components.

- *Hardware-Abstraction Layer (HAL)* provides generalized component interfaces. It shields the more general core functions from the specific and changing properties of underlying machine components. Internally, this layer contains the implementation of the component drivers.
- *Drivers* apply control instructions and configurations received from the control logic of higher building blocks of the operating system onto the components they are associated with and report components states in the reverse direction. Drivers connect to their associated components through their component interfaces. For asynchronous interaction, drivers are also connected with the interrupt system which allows drivers, as software system, to seize the processor resource for executing interrupt handling routines.

2.3.3 The Hardware Component Layer

Drivers interact with actual hardware components of the machine environment through a variety of mechanisms depending on the hardware architecture. Component interfaces can be implemented as ports, registers or via memory-mapped access. Interrupt signals

provide an important synchronization mechanism between machine components and the driver code as part of the operating system software. Processors are capable of processing interrupt signals received from the machine's interconnect fabric triggering the execution of a special interrupt cycle in order to reengage the operating system driver code.

The Hardware Component Layer is comprised of three main sub-layers:

- *Component Interfaces* – enable to program hardware components through ports, registers, memory mapped access, etc., depending on the processor architecture.
- *Components* – the elements that exit in the machine environment: processors, memory, MMU, controllers, network and other periphery interface cards, etc.
- *Interconnect Fabric* – links the components of the hardware environment with each other. The interconnect fabric does not only allow data transfer from one component to the other (e.g. from main memory to IO interfaces via fast bulk transfer), it also allows the transmission of control and synchronization signals which allow the coordination between the components and the operating system software. This mechanism is known as interrupt mechanism.

2.4 Summary and Discussion

This section presented a discourse in the history and evolution of operating system concepts. Concepts then have been categorized using an architectural framework of three layers: the application layer with applications and the application execution environment; the operating system layer with user management, process management and persistent data management, which are built on resource management and access to machine hardware components through a hardware abstraction layer and component drivers.

Some of the concepts presented in this section directly relate to the environment of a data center and its management. The following discussion will develop a generalization of operating system concepts which then can be mapped into the context of a data center.

The concept generalization has three dimensions:

- a *structural dimension* – includes generalized concepts relating to structure, which will later map into architectural structure of a DCI-OS;
- a *functional dimension* – includes generalized concepts relating to functionality a DCI-OS would need to provide; and
- an *organization dimension* – includes generalized concepts regarding organizational properties.

2.4.1 Generalization of Structural Concepts

Structural concepts primarily refer to the architecture of a system or environment. Concepts such as layers, modules and interfaces are examples of structural concepts. Purpose of structural concepts is to introduce order and patterns into a system or environment such that it can be extended, parts can be replaced and developed and maintained independently. All complex software systems should implement structural concepts. A DCI-OS is no exception.

Table 1 shows structural concepts which have been developed and have successfully been applied in operating systems.

Concept	Description
Layer	A *Layer* in a software system presents a structural concept which is internally comprised of a software system producing an execution environment of another software system residing in a layer above. A layer internally can have *Modules* as internal components and also *Sub-layers*. An operating system is a software layer residing between the hardware component layer and the application layer. A specific property of the operating system layer is that it manages the application layer above and the hardware component layer below. It thus forms the *management environment* for the two *managed environments* above and below.
Interface	Since layers form independent software systems, they are separated by interfaces through which elements from different layers can interact. An application, for example, can request more resources from the operating system by invoking a method of the operating system interface.
Application Layer	The Application Layer consists of two sub-layers: - the Application Environment and - the Application Execution Environment, The *Application Environment* contains all elements that constitute an application (application processes, application data, application configurations, etc). All elements can uniformly be seen as resources that are provided through the Application Execution Environment. Multiple Application Environments typically co-exist independently at the same time. Application-level resources can also be shared with other applications. --- The *Application Execution Environment* organizes the resource supply into the Application Environment, either based on explicit request or implicitly as needed by an application to proceed. The Application Execution Environment also mediates all interactions between the Application Environment and the Operating System, which includes: - access to application lifecycle operations (e.g. start, stop, etc.). - access to resource management (e.g. request, release resources), - access to the user management (e.g. login verification). The Application Execution Environment presents the resources abstractions as expected by the application and produced by the operating system. The Application Execution Environment includes the Interface with the operating system.

Operating System Layer	The Operating System coordinates the Application Environments transparently among each other making them execute in an environment of shared components. The Operating System organizes: - *User Management*, - *Process Management* and *Application Management*, - *Persistent Data Management*, All based on an abstraction of Resources produced and managed by an internal sub-layer: - *Resource Management*. --- Further internal sub-layers are: - *Hardware Abstraction Layer (HAL)* which resides between higher-ordered building blocks of Resource Management and the Device Driver layer. The HAL presents internal abstractions for basic component types found in the machine environment (e.g. block devices, character devices). - *Device Driver Layer* which implements the HAL interface and provides the basic access to the associated components of the underlying machine environment through their component interfaces. Device drivers fulfill three main tasks: - Accept control instructions received from resource management by applying control configurations to associated components. - Read data or status data from the component on request by higher components. - React to asynchronous signals received from the component (via interrupt mechanism) and signal in turn higher OS components.
Hardware Component Layer	The *Hardware Component Layer* is comprised of the computing elements of the machine environment. Hardware components have control interfaces through which they interact with the associated device drivers in the operating system. The Hardware Component Layer comprises three sub-layers: - *Component Interfaces* – enable to program hardware components through ports, registers, memory mapped access, etc., depending on the processor architecture. - *Components* – the bare elements that exit in the machine environment: processors, memory, MMU, controllers, network and other periphery interface cards, etc.

	- *Interconnect Fabric* – links the components of the hardware environment with each other. The interconnect fabric also provides the transmission of control and synchronization signals such as interrupt signals which allow the coordination between the components and the operating system software.

Table 1: Generalized operating system concepts in the structural dimension.

2.4.2 Generalization of Functional Concepts

Functional concepts refer to the functionality a system or environment should provide. Concepts such as application control or resource management are examples of functional concepts. Purpose of functional concepts is to define what a system or environment is supposed to deliver. All complex software systems should be implemented based on defined functionality.

Table 2 shows functional concepts used in operating systems.

Concept	Description
User Management	*User management* provides user identification and authentication. The concept of a User is fundamental to mechanisms of access control to applications and data, application control and resource use accounting. User management relies on credentials that are managed by the operating system allowing authenticating a user. Users can be grouped and assigned roles determining privileges to access applications. A special "super-" user role exists which has the privilege to access and manage the application execution environment and the operating system to set policy or perform administrative tasks in the system.
Process Management	*Process management* in an operating system provides control over processes as units of application execution. Process Management includes tasks of creating and controlling processes through basic lifecycle operations provide and exercising control over processes such as create, start, activate, pause, suspend, resume, interrupt (signal), stop and terminate. Isolation of processes and applications is another goal to allow co-existing applications and their processes to execute independently from one another. Inter-process communication is provided to support synchronization and communication among processes.
Persistent Data Management	Persistent data is core to applications. It is stored in non-volatile storage, often block device storage (disks, disk arrays, tape libraries). The operating system provides a basic organizational structure in form of resource abstractions of files and hierarchical file systems. Basic

	operations are defined: create, read and write files or directories. Besides the structure of a file system, the operating system does not interpret persistent data sets.
Resource Management	Resource Management includes two fundamental tasks: • The production of Resources in the quality (abstraction) and quantity (amount) that is expected by applications, and • The assignment of resources to applications. The first aspect relates to concepts of: - *Resource Abstraction* – a resource is either an elementary resource, which can directly be mapped onto an underlying machine component, or is a compound of resources assembled by the operating system in order to produce new desired resource properties. Resource abstractions are produced by resource transformation. - *Resource Topology* stands for an abstraction that describes an entirety of a resource set with their relationships. Applications not only need individual resources in order to proceed, they need a set of resources at once (all-or-nothing). - *Resource Transformation* in order to produce resources in desired abstractions, which can include a configured set (topology) of resources that are presented as a whole. - *Resource Generation* as the process of generating new resources with new properties and abstractions based on more elementary resources, eventually over multiple levels. - *Resource Virtualization* as the process of multiplying resources with same properties and abstractions for exclusive assignment to applications based on underlying shared resources. - *Resource Creation* as the process of mapping elementary resources onto underlying machine components. - *Protection* as process of enforcing access protection to resources and ensuring data integrity. The second aspect relates to concepts of: - *Scheduling* is the process of determining the assignment schedule over time of resources to underlying machine components. - *Resource Contention Policy* decides about resource assignment among competing processes. - *Accounting and Controlling Resource Use* to prevent monopolizing resource use in a shared environment.

Table 2: Generalized operating system concepts in the functional dimension.

2.4.3 Generalization of Organizational Concepts

Organizational concepts refer to the overall organization of a system, in particular how it relates to its environment. Examples of organizational concepts are time-horizons and defining behavior such as by setting policy the system should follow. All complex software systems should follow a set of defined organizational concepts.

Table 3 shows organizational concepts which have successfully been applied in operating systems.

Concept	Description
Comprehensive View of the Environment (Information Model – IM)	The ability any operating or management system to automatically manage an environment depends on the information this system has available. An operating system has a comprehensive view (meaning a full and accurate information set) of all layers surrounding it. The full set of information, maintained in form of internal data structures, allows the operating system to make management decisions. The entirety of information the operating system maintains about the environment and about itself is its *Information Model (IM)*.
Scope in time of the Information Model	An Information Model maintains information about a defined entity domain over a certain time horizon: • Extending into the past – memorizing prior states and conditions such as prior processor allocations to processes in order to implement a fair scheduling policy, • Capturing the presence – as accurately and consistently with the actual situation in the environment as possible, such as provided by instant processor response to interrupt signals allowing to mark state changes in components in the corresponding data structures instantly, • Extending into the future – such as predicting and foreseeing situations with regard to trends that may have been observed in the past and projected into the future, which forms the basis of the theory of working sets which is used for scheduling page replacements. Operating systems mainly work based on current information. They maintain little information about the past and incorporate only rarely information about the future. For instance, it cannot be "announced" to an operating system that a new application is scheduled to arrive next months. Resource reservation hence is not supported, which is, however, a requirement in a data center environment.

Scope in space of the Information Model	An Information Model also maintains information about a defined scope of elements it knows about. Elements known to an operating system include: • the higher-ordered application layer with application processes, application data and the states of these elements, • the underlying hardware component layer with the components and their states, and • the state about itself.
Policy	Policy is understood as a mean to express a goal an automated system should follow for making its decisions. Scheduling policy for resolving resource contention issues in an operating system may be to treat all application processes fairly or prefer higher prioritized processes and applications.
Sharing	The concept of Sharing occurs in two dimensions: - Sharing resources among applications for purposes of communication or collaboration. - Multiple application environments sharing a set of components in a data center.
Isolation	The concept of Isolation has two dimensions: - Avoid unwanted direct interference between application environments. Operating systems implement this concept by isolating address spaces and controlling visibility and access to resources associated with applications. - Allow control over unwanted indirect interference between applications that results from the shared environment in which they exist. In particular performance effects may result from interference in the underlying shared environment.

Table 3: Generalized operating system concepts in the organizational dimension.

Table 1 to Table 3 represented fundamental concepts from the domain of operating systems. These concepts will be reinterpreted for the context of a data center environment after the discussion of concepts from the domain of IT Management.

Chapter 3

Concepts from IT Management

3.1 Definitions

As ***IT Management*** is understood the *entirety of people, processes and systems that are needed for providing IT services* from strategic planning, design, transitioning into practice to operation and continuous improvement.

In contrast to operating systems, which mainly address the internal operation of computer systems, IT management also addresses the technical and organizational issues of the environment in which IT systems are used.

3.2 The Evolution of IT Management Concepts

One early root of IT management emerged with early computing systems. Systems required special skills to operate and also constant maintenance. Operators emerged as a profession of specially trained personnel who possessed those skills. The domain of system management has its roots in this tradition. With increasing volume of data being processed and stored, storage and data management later became specialized disciplines.

A second root of IT management is the domain of telecommunication and telecommunication networks, which have always been essential to enterprises with telephony and telegraphy services in the early days. Large telecommunication and later computer networks also required, and still require, specially trained personnel for operation and management.

With networking across sites and stronger integration between enterprises, the need for standards enabling more interoperability arose, which also reflected back to the management of related systems. A number of management standardization bodies emerged dedicated to telecommunications and IT management such as the Distributed Management Task Force (DMTF) [DMTF], the Telecommunications Management Network (TMN) [TMN] organization, or the Storage Networking Industry Association (SNIA) [SNIA] responsible for developing standards in the field of storage systems.

But also the more general standards' bodies such as International Organization for Standards (ISO), the Internet Engineering task Force (IETF) [IETF] or the Institute of Electrical and Electronics Engineers (IEEE) produced a number of standards related to IT

management. [Heg99] and [Slo94] are excellent sources for a comprehensive overview of the field of telecommunication and IT management.

A third dimension for IT management emerged with the need to manage the increasing number of operators, teams of operators and entire organizations concerned with the operation of large-scale computer systems or telecommunication networks. Their work needed to be organized and distributed and coordinated among organizations. Goals and metrics needed to be established for an IT organization and how IT organizations would fit into the overall organization of an enterprise.

The importance of the organizational (or "people") aspect of IT management has constantly been increasing. Practices from business management have been introduced into IT organizations defining how their business is run such as operate IT as a profit center producing a stream of revenue for IT services.

Due to the increasing reliance of businesses on IT, quality management in IT has become an important factor as well with attempts to adopt manufacturing practices for quality assurance for IT such as Six Sigma practices [Pan01].

More and more complex IT environment also led to the emergence of a more carefully considered lifecycle in IT, which begins early with strategic planning, design of IT services and their transition into IT systems as separate sets of tasks from the later lifecycle stages of operational management, which traditionally have been more in focus of IT management. Furthermore, it has been realized that IT management also does not end with maintaining IT operational. Constant improvement needs to be factored in as well. All these dimensions have been emerging over the years and have been factored into management frameworks and practices.

3.2.1 IT Services and IT Service Management

One particular problem IT management has been facing over the years is the gap between the IT world with its people, processes and systems and the business world, the world of users of IT. The problem manifests itself not only in IT management, it also relates to business process management and software engineering, but it is always IT that is pointed at when systems fail to provide the expected function.

Linking the business world (or world of users) better with the world of IT has become a major driver in IT management. On basis of a clear understanding of who is providing service to whom, IT and IT management has to "learn" to take the user's perspective rather than making users familiar with the complexities in IT.

IT Service Management (ITSM) has become the synonym for efforts managing information technology centered around the user's perspective and, in a business context, IT's contribution to the business in terms of its cost and value. ITSM stands in deliberate contrast to technology-centered approaches to IT management and business interaction: *"Providers of IT services can no longer afford to focus on technology and their internal organization, they now have to consider the quality of the services they provide and focus on the relationship with customers."*, [Bon02].

The IT Service Management Forum [itSMF] is an independent and internationally-recognized forum for IT Service Management professionals worldwide.

With the goal of more focus on the business context in which IT is managed, IT service providers have developed their own interpretations and product and service portfolios.

Hewlett-Packard's idea of an *Adaptive Enterprise* targets Business Agility [HPBA03] as main differentiator by allowing customers to regain control over IT and take advantage in their markets from an IT environment that is more responsive and easier to adapt to change in business [HPAEM03]. Transforming IT systems from a factor of limiting change in business to a factor that is actually fostering change and agility in business requires substantially higher interoperability, looser coupling of components into services and automation. And since business agility is seen as a differentiator for customers competing in their markets, it is also a differentiator for IT system and solution vendors providing those capabilities [IDC03].

IBM's programs in this realm have been called On Demand computing and Autonomic Computing [Kep03], primarily aiming at higher returns on IT investments by automating management, but also addressing concerns of increasing scale and complexity in IT.

Key element of the interface between users and providers of IT is the *IT Service*, which is a contractual relationship between IT service provider and consumer specifying the services and service levels in form of an agreement. IT Services are expressed in consumers' terms, such as "email service", with expected business objectives regarding capacity, security or availability. IT interfaces with its users through these contractual relationships and also derives its directives through those relationships [Bart04].

3.2.2 IT Management Frameworks

A number of IT frameworks exist for IT service management. ITIL is chosen for a more detailed discussion because of its widespread use.

The *IT Infrastructure Library (ITIL)* [ITIL] is a set of best practice guidance for IT Service Management. ITIL is owned by UK's Office of Government Commerce (OGC) and consists of a series of publications giving guidance on the provision of quality IT Services, and on the processes and facilities needed to support them.

The IT Infrastructure Library framework version 3, for example, incorporates a comprehensive lifecycle view for IT Services over five major stages:

- *Service Strategy* [Iqb07],
- *Service Design* [Rud07],
- *Service Transition* [Lac07],
- *Service Operation* [Can07], and
- *Continual Service Improvement* [Spa07].

Recent trends focus on how IT functions can be provided, consumed and managed more uniformly as services that are closer linked to the business needs they support. Business-driven IT Management (BDIM) is one of those trends. Another related trend is to more explicitly and transparently manage processes as the fundamental link to the business world. Business Process Management (BPM) [Wes07] is a practice of attempting to constantly improve business processes, which relates back to IT management by the pattern of organizing not only the core business functions as defined and predictable processes, but also IT management itself.

In order to summarize common experiences to better organize IT management, a number of frameworks for IT management have been established over the years. Two prominent

representatives are the IT Infrastructure Library for general IT management, and Tele-Management Forum's *enhanced Telecom Operations Map (eTOM)* [eTOM] and *New Generation Operations Systems and Software (NGOSS)* [NGOSS] which originally have been addressing the management of telecommunications networks but later, with the convergence of telecommunication networks into general communication networks used in IT, they have been broadening their scope towards general IT management reaching up into the business process layer.

Both frameworks describe the organization of IT management in terms of processes performed over the lifecycle of IT services, from early, pre-operational stages of strategy, planning and definition to operational stages of service delivery (or fulfillment, assurance) and service support. [Scha07] provides a comparison between ITIL and NGOSS under the view of business-driven service level management.

ISO/IEC 20000 [ISO2K] is another framework for IT management released by ISO. The Control Objectives for Information and related Technology (*COBIT*) [COB] is another.

Despite the different origins and flavors of these management frameworks, they all seem to be converging around notions of *services* and *processes* and their *lifecycles*.

The following discussion will refer to the IT Infrastructure Library. ITIL has been chosen mainly because of the research background and Hewlett-Packard's strong position in IT management products and services. ITIL thus had been considered most relevant, although another IT management framework could have been chosen as well.

3.3 Categorization of IT Management Concepts

For purpose of categorization, some further definitions from ITIL are provided, which will be used in the categorization shown in Figure 2.

- As *IT Service* (in ITIL) is understood a service provided to one or more internal or external customers by an IT Service Provider. An IT Service is based on the use of Information Technology and supports the customer's business processes. An IT Service is made up from a combination of people, processes and technology and should be defined in a Service Level Agreement. An *IT Service Provider* is an organizational entity that provides IT Services to internal or external customers.
- A *Service Level Agreement (SLA)* is a part of a service contract where the level of service is formally defined. In practice, the term SLA is sometimes used to refer to the contracted delivery time (of the service) or a performance-related metric.
- As *IT Management* or *IT Service Management* is understood the implementation and management of quality IT Services that meet the needs of the business. IT Service Management is performed by IT Service Providers through an appropriate mix of people, process and Information Technology.
- As *IT Operations* (Service Operation) is understood the activities carried out by IT operations control, including console management, job scheduling, backup and restore, and print and output management. IT Operations is also used as a synonym for Service Operation. As *IT Operations Management* is understood the function within an IT Service Provider that performs the daily activities needed to manage IT Services and the supporting IT infrastructure. IT Operations Management includes IT Operations Control and Facilities Management.

Application Layer Application Environment – Managed Side			**IT Management Environment** Management of the Application Environment
Role User Consultant	**Business Services** (e.g. payroll, supply chain, order processing, fulfillment). Business Logic & Processes		**Business Service Management** (e.g. customer management, user support), Business Process Management
IT Architect (planning, realization), IT organization (deployment, operation)	**IT Application Services** (e.g. SAP ERP, Oracle CRM) Topology of customized and operational **Application Instances**.		**IT Service Management** with **- Service Support:** service request management, incident management, problem management, configuration and change management, release management, and software asset management. **- Service Delivery:** service level management, capacity management, IT service continuity management, availability management, financial management.
Application Execution Environment – Managed Side			**Management of the Application Execution Environment**
IT Operator IT organization (infrastructure planning and operation)	**Application Services Execution Environment** Topology of operational **Infrastructure Services** (*operational* application software and data on a topology of operational infrastructure resources).		**IT Infrastructure Service Management** Goal is to provide and manage: **- Infrastructure Management Services** for producing Infrastructure Resources, infrastructure planning and design, deployment management, operational management with aspects of: - *Grounding Requirements* (refinement), - *IT Design* and *Topology Models*, - *Provisioning:* plan, request, allocate and assign resources (current and future), - *Deployment:* apply configurations, - *Operational Management*.
	Topology of **Infrastructure Resources** (*configured* networks, servers, storage that are assigned to an application).		
Data Center Component Layer			**Data Center Component Management**
IT Operator IT organization (infrastructure planning and operation)	**Data Center Components** Component Interfaces		**Data Center Component Management** with component interfaces for operators (console, GUI) and management systems (SNMP, WBEM, WS-Management).
	Components: bare servers, storage, devices (switches, routers, firewalls, balancers).		Component-level Management: read status, deliver monitoring data, control, apply configuration.
	Interconnect fabrics: LAN, SAN, WAN uplinks.		Access to Component Management Interfaces.

Figure 2: Categorization of IT Management concepts.

- As *IT Infrastructure* is understood the entirety of the hardware, software, networks, facilities, etc., that are required to develop, test, deliver, monitor, control or support IT Services. The term IT infrastructure includes all of the Information Technology but not the associated people, Processes and documentation.
- As *IT Infrastructure Management* is understood a part of IT Operations Management which deals with planning, designing, configuring and maintaining *IT Infrastructure components*. As *IT Infrastructure Component* is understood a technically enclosed hardware system that is located within the environment of a data center.

Given that the domain of concern of this thesis is IT Infrastructure automation in the data center, emphasis will be IT Service Management and IT Infrastructure Management.

Like in the previous section on categorizing concepts from operating systems, a methodology of layers will be applied.

Figure 2 shows a categorization of IT Management Concepts. IT management occurs as a second stack of layers with each layer on the **Management Side** (shown on the right in Figure 2) accompanying components of an associated layer of the **Managed Side** (shown on the left). The local operating system layer still exists between the application environment and the local data center components. However, this layer is not involved in data center management tasks. It is responsible for managing local machine resources and application processes local to a machine. The local operating system layer is not aware of the broader context of the data center within which machines and applications reside.

IT Management exists as a separate environment of hardware, software and people on the managed side. Layers in the managed environment are accompanied by layers in the management environment.

In order to relate concepts from operating systems to IT management, the three layers shown in Figure 2: Application Layer, Operating System Layer and Hardware Component Layer need to be reinterpreted for the environment of IT management:

- The concept of the **Application Layer** refers to the actual applications running in a data center. It consists of the
  - *Application Environment* with Business Services, IT Application Services and operational Application Instance and the
  - *Application Execution Environment*, which includes operational Infrastructure Services residing on operational Infrastructure Resources with the Local Operating System Layer residing between them.
- The concept of the Operating System Layer responsible for managing an environment is generalized into the **IT Management Environment**. The three layers of the managed environment are accompanied with three layers of the Management Environment:
  - *Management of the Application Environment* with Business Service Management, which includes Business Process Management, as well as IT Service Management with Service Support and Service Delivery.

  - *Management of the Application Execution Environment* with IT Infrastructure Service Management including planning and design, FCAPS management, technical support. A number of further tasks are included here such as grounding requirements, IT design and topology models, provisioning, deployment and operational management.
  - *Data Center Component Management* including Component Interfaces and Component-level Management.
- The concept of The Hardware Component Layer refers to the physical IT hardware components found as in a data center comprising the **Data Center Component Layer**. It consists of
  - Component Interfaces and
  - Components connected through
  - Interconnect Fabrics.

3.3.1 The Application Layer

3.3.1.1 The Application Environment

On the managed side, the application layer shown in the upper left box in Figure 2 is formed by concept of:

- *Business Services* perform business functions such as payroll or supply chain functions. They rely on IT Application Services that are connected through properly configured business logic or business processes.
- *IT Application Services* provide functions for business services, typically provided by standard applications such as SAP or Oracle. Those Application Services consist of a number of:
- *Application Instances*, which are local instances executing on local systems in a data center. Application instances form the smallest unit of execution in the Application Environment.

Business services support business users, people who conduct some business functions. These people use the business services. Business logic or business processes implement business services. They are planned, designed and implemented by various people engaged as consultants or developers. Both roles need to be supported in this category.

The underlying services shown in the middle left box in the figure refer to IT Application Services are services provided by running, customized applications, which are often standard applications such as SAP or Oracle in enterprise environments. Customized applications implement the business logic or business processes. The combination of IT Application Services plus customized Business Logic and Business Processes results in the desired Business Service at the top layer.

Since IT Application Services themselves can be of a complex nature, a further break down into Application Instances is often necessary. IT Application Services are comprised of instances of running applications that are needed for an IT Application Service. For example, SAP ERP follows a multi-tiered architecture consisting of a NetWeaver Application Server (one application instance), a database (a second

application instance) and eventual further modules such as Business Information Warehouse or Master Data Management. All these application instances together provide the overall application service of a SAP ERP system. All participating application instances must be properly customized, configured and made operational in order to provide their contributions to an application service.

3.3.1.2 The Application Execution Environment

On the managed side, the Application Execution Environment shown in the middle left box in Figure 2 is comprised of:

- Topology of operational *Infrastructure Services*. As operational Infrastructure Service is understood as operational application software and data installed on operational infrastructure.
- Topology of *Infrastructure Resources*. As Infrastructure Resources are understood configured networks, servers, storage that provide the operational environment for Infrastructure Services.

The *Application Execution Environment* consists of a topology of configured hardware and software components that is capable of supporting all Application Instances of an Application Service.

Operational *Infrastructure Services* are produced when bare (dead) data center components are selected from the inventory and turned into operational (alive) components by properly configuring them, which includes installing the proper software on them (application code and data). The entirety of these operational infrastructure services forms the application services execution environment. Application instances are the basic units mapped onto infrastructure services. Typically, a physical or virtual server is provisioned for an application instance. Proper configuration of this server means that the server must have access to disks with proper software and data configured as well as to networks

Infrastructure Resources are properly configured and operational infrastructure components onto which application data and customizations can be deployed. The difference between infrastructure resources and infrastructure services is the presence of application-specific configurations including application software and data on them. Infrastructure resources are created from data center components such as networks, storage and server components.

3.3.2 The IT Management Layer

3.3.2.1 Management of the Application Environment

On the management side of the Application Environment, the concepts of the application layer are accompanied by respective layers of management concepts shown in the upper right box in the figure:

- *Business Service Management* (e.g. customer management). Business Service Management includes Business Process Management (BPM) [Wes07], which aims at constantly adapting the business processes to changing business conditions to improve business results. The Capability Maturity Model Integration (CMMI) is a practice with a process improvement approach that provides

organizations with the essential elements of effective processes. Management at the business service or business process layers is inherently people-oriented.

- *IT Service Management* with *Service Support* (service request management, incident management, problem management, configuration and change management, release management, and software asset management) and *Service Delivery* (service level management, capacity management, IT service continuity management, availability management, and financial management).

IT Service Management generally describes the discipline of managing IT services. It includes a broad range of management practices targeting IT management organizations to structure their work. A number of frameworks exist such as ITIL. Other frameworks include the ISO/IEC 20000 [ISO2K], Microsoft Operations Framework (MOF) [MOF], the Control Objectives for Information and Related Technology [COBIT], IBM's Process Reference Model [PRM-IT] or ISO 9001 [ISO9K].

The two major categories of service operation for ITIL are:

- *Service Support* includes: service request management, incident management, problem management, configuration and change management, release management and software asset management.
- *Service Delivery* includes: service level management, capacity management, IT service continuity management, availability management and financial management of IT services.

Further areas may need to be addressed such as security and compliance management, which are not discussed here.

3.3.2.2 Management of the Application Execution Environment

On the management side of the Application Execution Environment, the middle layer shown in the middle right box in Figure 2 is formed by concepts of:

- *IT Infrastructure Service Management* with infrastructure planning and design, deployment management, operations management (fault, capacity, availability, performance, security), technical support.
- Managed *Infrastructure Services*.
  - Grounding Requirements (refinement).
  - IT Design and Topology Models.
- Managed *Infrastructure Resources*
  - Provisioning: plan, request, allocate and assign resources (current and future).
  - Deployment: apply configurations.
  - Operational Management.

Goal of *IT Infrastructure Management* is to provide operational infrastructure services considering all aspects of: infrastructure planning and design, deployment management, operations management (fault, capacity, availability, performance, security) and technical support.

For Infrastructure Services, concepts of grounding and topology are important, which are explained in more detail later.

Related aspects of infrastructure management include requirement acquisition and *Grounding* requirements, which is a concept of refining requirements from higher terms into more specific terms used for data center planning and design. *Requirements* for new business services must be translated into more specific requirements for capacities of servers, storage and networks. Workloads must be estimated. Grounding requirements from business into IT terms relies on expertise of experienced IT specialists.

The concept of an *IT Design* is important since the system to be built does not exist, yet. Many management systems fall short of supporting design stages. They focus on operational management where the managed system is already in existence. For this reason, designs are often made outside of management systems.

Information Models are used to define designs (in form of topological, structural, capacity, performance and security models). A design guides subsequent stages of acquisition, configuration and deployment of IT components when the designed system is brought into existence in a data center. Standards for information models exist such as the Common Information Model [CIM], in which designs can be formally represented.

Since individual instances are not sufficient for creating operational application services, their relationships must be represented in designs as well. A concept of a *Topology* is introduced as a graph which includes all components at a respective layer along with their relationships. The concept of a topology can be generally applied at any layer, from application services consisting of a topology of configured application instances to infrastructure services consisting of a topology of configured infrastructure resources.

For Operational Infrastructure Components, concepts of provisioning and deployment are important.

Provisioning is the process of making resources available for use. It includes stages of requesting resources (e.g. as result of grounding higher-ordered requirements), allocating resource capacity and assigning particular resource instances to a request. In contrast to operating systems, time scales of requests and provisions of resources are much longer (may be years). Hence, planning and managing "future" resources is essential. Future resources may not exist at the time when they are requested, yet the information that they will be available at some future time is already known and can be factored into planning processes. An operating system only knows the resources that currently exist in the machine environment. Most IT management tools also only can deal with current inventory, which is a significant limitation for data center management. Resource management for a data center must incorporate future resource planning with anticipated demands such that resources are ready in sufficient quantity when needed.

Deployment refers to the process of applying proper configurations to components such that they become operational for the desired purpose. This includes transfer of data such as application data into software components, application programs onto servers or configuration data into hardware infrastructure components.

3.3.2.3 Data Center Component Management

Component-level management assumes access to components for management. This can be provided in form of user interfaces for operators (management consoles) or in form of programmable interfaces. One important requirement for an operating system is to have programmatic access to data center components through API. A number of standards

exist for programmatic access to management interfaces: SNMP, WBEM or more recently Web Services Management.

- Component-level management primarily refers to:
- read status information from components,
- deliver monitoring data from components,
- issue control instructions to components and
- apply configuration data onto components.

At the lowest level of management, access to management interfaces must be provided through basic network interconnects.

3.3.3 The Data Center Component Layer

Data Center Components include everything in a data center which is physically connected and can be used to create infrastructure components and services. It includes bare servers and storage as well as devices such as switches, routers, firewalls or load balancers. All these components are wired through interconnect fabrics for LAN and SAN as well as LAN/WAN for management networks (if a separate network is used).

Three main categories of Infrastructure Components exist in the data center:

- Data Center Components with Component Interfaces,
- Components: bare servers, storage, devices (switches, routers, firewalls, balancers), and
- Interconnect fabrics: LAN, SAN, WAN uplinks.

3.4 Scope of the Data Center Infrastructure Operating System (DCI-OS)

Considering the broad scope of IT Management, from Business Process Management to Data Center Component Management, it is clear that the scope of a DCI-OS must be focused on aspects that are related to infrastructure and its management. A DCI-OS is situated in the management environment and consequently part of the right column in the figure. The box in the right column in Figure 3 shows the areas addressed by the DCI-OS.

The scope of a DCI-OS addresses two main areas of concern:

- The area of *Managed Infrastructure Services* on *Managed Infrastructure Resources* and
- The area of *Data Center Component Management*.

Infrastructure Services provide the basic abstraction offered in the Application Execution Environment. These Infrastructure Services are comprised of operational application software and data on operational infrastructure resources. An example is a properly configured server resource with installed application software and data. Infrastructure Resources in this example are the server resource which is properly connected to the right network and storage resources, which are also Infrastructure Resources needed in the Application Execution Environment. Infrastructure Services and their underlying topology of Infrastructure Resources are part of the managed side (left column in Figure 3) and must be complemented by management (right column in Figure 3), which is part

of the DCI-OS. The DCI-OS fulfills management tasks for Infrastructure Services and Infrastructure Resources as well as bringing them together in the right setting of configurations and relationships, which is referred to as *Topology*. This domain of a DCI-OS is concerned with the context of an Application Execution Environment, and multiple of those environments that co-exist in a data center. It represents the "application context" that must be maintained in a DCI-OS, like an operating system must maintain the context of the applications executing on them. Likewise an operating system, the DCI-OS is not aware of the meaning of actual application logic or application data. It only supplies the appropriate Infrastructure Services into the Infrastructure Execution Environment such that application systems can execute.

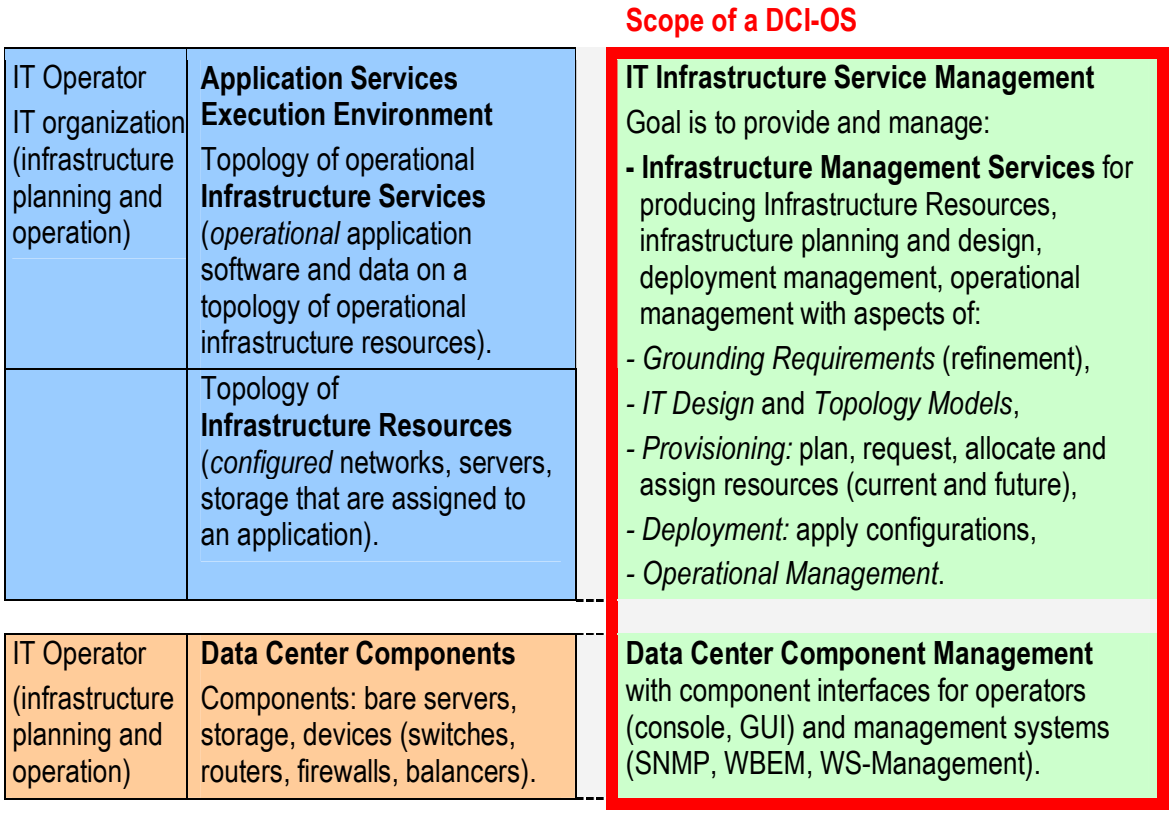


Figure 3: Scope of the Data Center Infrastructure Operating System (DCI-OS).

The other area addressed by a DCI-OS is the environment of the data center in which it exists. Components exist in the data center on which Infrastructure Resources can be created by applying proper configurations. Since components typically occur in numbers, they can be considered and treated as *Pools*. A Pool is a set of Data Center Component or created Infrastructure Resource instances of the same type. As Infrastructure Resources are needed for Infrastructure Services, capacity is assigned from Pools. In a pooled environment is not significant, which component or resource instances are assigned for a particular demand. All instances in a pool are considered equal. If they are not equal, different Pools must be maintained.

The presence of further concepts in Figure 3 of Grounding Requirements, IT Design and Topology Models, Provisioning, Deployment and Operational Management indicates

functionality a DCI-OS must provide in order to deliver the Infrastructure Resources and Services. Those will be discussed in the following chapters.

3.5 Summary and Discussion

This section provided an overview of concepts from the domain of IT Management and categorizing them based on a similar architectural view used in the previous section for categorizing concepts from the domain of operating systems.

Figure 2 showed the main concepts categorized into an architectural framework for IT management using three layers:

- the *Application Layer* with Business Services managed by Business Service Management and underlying IT Application Services managed by IT Service Management.
- the *IT Management Environment* as vertical stack of layers of management abstractions associated with layers of managed abstractions.
- the *Data Center Component Layer* managed by Data Center Component Management.

While the Separation into three layers is similar to the three layers discussed for the domain of operating systems, the vertical split into a managed and a management side is different. The operating system layer provides the management for the applications and the hardware component systems of a machine. It resides a as *layer between* the application and the hardware component layers in the *same environment* of the machine.

From the categorization of IT management concepts, the scope of a DCI-OS has been defined related to the area of Infrastructure Services provided on a Topology of Infrastructure Resources. Another area of concern for a DCI-OS is the environment of Data Center Components. Both areas contain their elements that need to be managed by a DCI-OS. The two areas of concern reflect the application context on one hand and also the context of a data center on the other hand. This is similar to an operating system, which also brings the contexts of applications together with the context of the components of a machine environment.

The following chapter will further correlate concepts from both domains of operating systems and IT management and condense them into requirements for a DCI-OS, based on which the architecture of such a system then will be developed.

Similarly like in Chapter 2 where concepts form operating systems were categorized based on three dimensions, these dimensions are applied here as well to categorize concepts from the IT management domain:

- a *Structural Dimension* – includes generalized concepts relating to the structure, which will later map into architectural structure of a DCI-OS;
- a *Functional Dimension* – includes generalized concepts relating to the functionality a DCI-OS would need to provide; and
- an *Organization Dimension* – includes generalized concepts regarding organizational properties.

The approach in this discussion is to take the generalized concepts from operating systems discussed in Chapter 2 and interpret them for the domain of IT management.

Eventual additional concepts are added, which are new in the domain of IT management and do not exist, or exist in different kind, in the domain of operating systems.

3.5.1 Concept Generalization in the Structural Dimension

Structural concepts primarily refer to the architecture of a system or environment. Concepts such as layers, modules and interfaces are examples of structural concepts. Purpose of discussing structural concepts from operating system in context of data center management is to derive structural requirements for a DCI-OS.

Table 1 shows the structural concepts which have been discussed for operating systems in Chapter 2 and how they are mapped into the domain of data center management.

Concept	Description
Layer	As layers occur in the managed domain, they also occur in the management domain. Figure 2 presented the Application Layer with associated Business Services Management and IT Service Management, the Layer of the Application Services Execution Environment with Infrastructure Services and the associated IT Infrastructure Service Management and the Data Center Component Layer with the associated Data Center Component Management Layer. A layer internally can have *Modules* as internal components and also *Sub-layers*. In contrast to an operating system, management is not residing between the hardware component layer and the application layer. It exists as separate stack of management layers. Similarly like an operating system, the management layer manages the application layer and the hardware component layer in a data center.
Interface	Since the managed environment and the management environment form independent systems, they are separated by management interfaces through which elements in both environments can interact. An application, for example, can request more resources from the management system by invoking a method of the management interfaces. The management system also can request the status of a management component through its management interface. A difference to operating system interfaces is that management interfaces are not unified. Not one management interface exists; many exist for each managed component and management system. Communication with management interfaces today occurs as remote invocations mediated through some middleware. A number of standards have been defined for management API. Since in contrast to operating systems, management systems rely on human operators for making decisions. For this purpose, user interfaces (consoles) to management systems are important.

Application Layer	The Application Layer also consists of two sub-layers: - the *Application Services Environment* and - the *Application Services Execution Environment*, The *Application Services Environment* in a data center environment, maps to systems of IT application services performing coordinated business functions such as SAP ERP (Enterprise Resource Planning) or Oracle CRM (Customer Relationship Management). *IT Application Services* consist of a number of interacting application processes executing on a set of machines which are configured in a way such that they support the interaction. Application Data is a key part of Application Services. There is a spectrum of processes occurring in an application environment of a data center, ranging from operating system processes executing on individual machines to workflows and business processes across application instances. Multiple Application Service Environments typically co-exist independently at the same time in a data center. Application-level resources can also be shared with other applications. _____ The *Application Services Execution Environment* organizes the resource supply into the Application Environment, either based on explicit request or implicitly as needed by an application to proceed. In a data center, the Application Services Execution Environment does not exist in form of an abstract machine per se. Application components execute directly on sets of machines with local operating systems. The Application Services Execution Environment of a data center is provided by its management functions and processes that allow applications to operate. It is thus today mainly provided by operators in data centers who carry out basic functions of resource management (e.g. capacity planning and adjustments, monitoring and observation, etc.). The Application Services Execution Environment provides means to configure applications and connect them to their application data. The Application Services Execution Environment provides means to retrieve and manipulate state of the application and application data such as needed for backup, archive or migration purposes. The Application Services Execution Environment provides the interface to the associated operating or management systems to request and release resources and control to the application lifecycle (create, start, stop, suspend, resume, etc.). The Application Execution Environment also allows creating and presenting the resources abstractions as expected by the application. The Application Execution Environment includes the Interface with the management system.

Management Layers	In contrast to the Operating System Layer, layers of management accompany the layers of the managed environment. Management forms a separate stack of layers (see Figure 2). For a DCI-OS, two management layers are of interest: • *IT Infrastructure Service Management* with o *Managed Infrastructure Services* and o *Managed Infrastructure Resources*; and • *Data Center Component Management* with accessing data center components through their management interfaces. Since infrastructure services and resources are directly provided and associated with the application services, they represent the "application view" a DCI-OS has; while the data center component management represents the "data center view". This duality corresponds to operating systems which also have views into the application domain and the domain of machine components. Standard building blocks of an operating system such as user management, process management, application management and persistent data management are generalized into specific Infrastructure Services the management environment can provide. Examples of Infrastructure Services are: - *User Management Service*, e.g. Microsoft Active Directory, - *Application Lifecycle Management* (as generalization of process and application management) with aspects of: - *Service Design,* - *Provisioning,* - *Deployment* and - *Operational Management*. - *Data Management Services*, e.g. for backup and archiving (as generalization of persistent data management). These building blocks are implemented and provided as Infrastructure Services, which means they are active software components as part of the management system. Basis for these higher-ordered Infrastructure Services is like in an operating system: - *Resource Management*, which is generalized to a concept of Infrastructure Resources which are organized as Topologies in order to provide the resources into the Application Services Execution Environment as needed.

	Further internal sub-layers can be seen similar to an operating system: - *Hardware Component Abstraction Layer (HCAL)* which resides between higher-ordered building blocks of Resource Management and the Device Driver layer. The HAL presents internal abstractions for basic component types found in the data center environment (e.g. routers, switches, servers with their configuration systems, storage with their configuration systems). - *Hardware Component Driver Layer* implements the HAL interface and provides the basic access to the associated components of the data center environment through management interfaces. It corresponds to the device driver layer in an operating system. Device drivers fulfill three main tasks: - Accept control instructions received from resource management by applying control configurations to associated components. - Read data or status data from the component on request. - React to asynchronous signals received from the component (via interrupt mechanism) and signal higher-ordered components.
Data Center Component Layer (~Hardware Component Layer in OS)	The *Data Center Component Layer* corresponds to the Machine Component Layer in operating systems. It is comprised of the computing elements of the data center environment. Data center components have management interfaces through which they interact with the associated drivers in management systems. Interactions with component management interfaces can occur for operators through consoles and user interfaces. Interactions with management systems occur through management protocols such as SNMP, WBEM or WS-Management which is mediated through a management network interconnect, which may exist as a separate network or is part of the regular network. Management protocols facilitate access to status information, to set new configurations and to interact asynchronously via traps or asynchronous event messages which allow the management software to synchronize with the managed component.

Table 4: Generalized IT Management concepts in the structural dimension.

Conclusions for structural requirements for a DCI-OS are:

- A DCI-OS has two main parts reflecting the application and the data center view:
  - *IT Infrastructure Service Management*
    - Managed Infrastructure Services and
    - Managed Infrastructure Resources; and
  - *Data Center Component Management*.

- Internal layers comprise:
  - Infrastructure Management Services layer – providing the actual management functionality,
  - Resource Management layer – producing resource abstractions needed in the application environment as well as managing the components of the data center environment as resources,
  - Hardware Component Abstraction Layer (HCAL) – providing a uniform access layer to data center components for Resource Management,
  - Hardware Component Drivers – for accessing data center components via their control interfaces.
- Functional building blocks are implemented as application-level services operating on machines that are dedicated to management.
- Interfaces through standard middleware protocols allow for the interaction among the infrastructure management services.
- A modular structure should support reuse and extensibility. Choosing a structural building block of an Infrastructure Service for a DCI-OS supports modularity.

3.5.2 Concept Generalization in the Functional Dimension

Functional concepts refer to the functionality a management system. Purpose of discussing functional concepts is to derive functional requirements of a DCI-OS.

Table 2 shows functional concepts which have been discussed for operating systems in Chapter 2 and how they are mapped into the domain of data center management.

Concept	Description
User Management	*User management* provides user identification and authentication for applications and management. In contrast to operating systems, which manage local users, user management in data centers of often fragmented and often also part of applications. A number of systems exist that can be used as Infrastructure Management Services for user management in a data center. With regard to a DCI-OS, user management is not addressed directly by implementing own functionality. It can be incorporated as one its Infrastructure Management Services.
Application Service Lifecycle Management (~Process Management in OS)	*Application Service Lifecycle Management* is an important function of data center management. It includes aspects of • *Application Service Design* – includes the specification of requirements (functional, non-functional), and further into o *Grounding* of those requirements into a set of specifications for infrastructure resources and services that can be produced in a data center, o *Development of IT Design and Topology Models* which provide the blueprints for the Infrastructure Services

	and underlying Infrastructure Resources which need to be produced for the application service. • *Provisioning* – comprising the planning, request, allocation, assignment of Infrastructure Resources for the Infrastructure Services needed for an Application Service. • *Deployment* – application of configurations derived from design blueprints to Infrastructure Resources and Hardware Components in order to produce the desired Infrastructure Services needed for an Application Service. • *Operational Management* – to manage the entirety of Hardware Components participating in an Application Service, upon which Infrastructure Resources have been built and delivered into the Application Execution Environment. *Task Automation* of these management functions is a goal for DCI-OS.
Application Data Management (~Persistent Data Management in OS)	Application Data are key assets in IT. Persistent data sets of IT Application Services are stored in special applications such as database or warehouse applications, which themselves can be IT Application Services. Since data maintained for IT Application Services often is business-critical, special attention is given to availability, reliability, security, access and compliance of data stored and exchanged. There is a spectrum of tiers where application data can reside in a data center (block device storage, central storage and file servers, storage arrays). Storage management is a key discipline in data center management. Security, availability, integrity, and regulatory requirements (such as audit of access to data) are important dimensions of application data management, which is also a key discipline in data center management. Access control, data replication, fail-over, backup and recovery are essential building blocks of data management. While application data management concentrates on the aspects of pure data (with content unknown), information management focuses on the lifecycle, distribution and aggregation of information (content is know). Both are key disciplines in data center management. Most applications provide their own capabilities for application data management, which is in contrast to operating system. Applications only use persistent storage from the data center environment. Consequently, with regard to a DCI-OS, direct application data management is not part of a DCI-OS. What is in scope of a DCI-OS is to produce and manage the needed storage as Infrastructure Resources for application data management. The Application Execution Environment provides means to configure applications and connect them to their application data. Common functionality such as backup or archival can also be supported as higher-ordered Infrastructure Management Services.

Resource Management (~Resource Management in OS)	Resource Management in operating systems included two fundamental tasks (see Chapter 2): • The production of Resources in the quality (abstraction) and quantity (amount) that is expected by applications, and • The assignment of resources to applications. These general aspects apply to a data center environment. Although the kind and granularity of resources differs between a machine environment and a data center environment, the principal pattern applies: *a resource is produced by applying a configuration to a machine component making it usable by an application*. For this reason, the concepts that have been identified for resources in operating system, fully apply to a DCI-OS: - *Resource Abstraction* – a resource is either an elementary resource, which can directly be mapped onto an underlying data center component, or is a compound of resources assembled by the DCI-OS in order to produce new desired resource properties. - *Resource Topology* stands for an abstraction that describes an entirety of a resource set with their relationships. Applications not only need individual resources in order to proceed, they need a set of resources at once (all-or-nothing). - *Resource Transformation* in order to produce resources in desired abstractions, which can include a configured set (topology) of resources that are presented as a whole. - *Resource Generation* as the process of generating new resources with new properties and abstractions based on more elementary resources, eventually over multiple levels. - *Resource Virtualization* as the process of multiplying resources with same properties and abstractions for exclusive assignment to applications based on underlying shared resources. - *Resource Creation* as the process of mapping elementary resources onto underlying data center components. - *Protection* as process of enforcing access protection to resources and ensuring data integrity. Also the concepts relating to Scheduling, Resource Contention Policy and Accounting and Controlling Resource Use apply: - *Scheduling* is the process of determining the assignment schedule over time of resources to underlying data center components. - *Resource Contention Policy* decides about resource assignment among competing processes. - *Accounting and Controlling Resource Use* to prevent monopolizing resource use in a shared data center environment.

Table 5: Generalized IT Management concepts in the functional dimension.

Conclusions for functional requirements for a DCI-OS are:

- The main tasks of an operating system: application and process management relate in context of a DCI-OS to an *Application Service* for which the *Application Service Execution Environment* with all needed Infrastructure Services and Infrastructure Resources must be provided.
- *Production of Infrastructure Services and Infrastructure Resources* for Application Service is a core function of a DCI-OS.
- *Management Tasks* also apply to the concept of an Application Service in form of its Application Service Lifecycle Management with tasks of:
    - *Application Service Design* with Grounding and Development of IT Design and Topology Models,
    - *Provisioning*,
    - *Deployment*, and
    - *Operational Management*.
- *Task Automation* for management functionality is an important goal of a DCI-OS. (section 8.1 "The Task Automation Controller" provides a detailed design and implementation of task automation for a DCI-OS).
- *Protection* of resources with regard to controlling access or isolation must be supported by a DCI-OS in order to avoid unwanted interferences among Application Services which coexist in a shared data center environment.
- The concept of a Resource and its management functions in an operating system has been generalized into a *Topology of Infrastructure Resources*. An Infrastructure Resource is a Resource Abstraction needed by an Application Services. Since Infrastructure Resources can only be used in combination with other needed Infrastructure Resources (all-or-nothing aspect), a set of Infrastructure Resources is brought into context by a Topology.
- *Production of Infrastructure Resources* is an essential function of a DCI-OS which is achieved by applying configurations to data center components or recursively to underlying previously produced Infrastructure Resources and thus making them usable for an Application Service.

  Following qualitative categories of resource productions are distinguished:
    - *Resource Transformation* with sub-categories of
        - *Resource Generation*,
        - *Resource Virtualization*,
    - *Resource Creation*.

  Production of Infrastructure Resources also has a quantitative dimension.
- *Scheduling* is the process of coordinating resources that are shared among Application Services. Scheduling is an important function of a DCI-OS. Scheduling also implements the *Resource Contention Policy*.
- *Accounting and Controlling Resource Use* through monitoring and associating monitored data with the contexts of Application Services is a further task of a DCI-OS.

Functionality that is missing or only present in rudimentary form in operating systems must be supplemented to functional requirements for a DCI-OS. This missing functionality primarily relates to aspects of:

- *Data Center Planning, Design and Management*.

That these aspects are not considered in operating systems reflects the fact that data centers are not static environments like the environment of a machine, which hardly changes over its lifetime. In a data center, each data center component goes through a lifecycle causing constant replacement of inventory.

This aspect must be factored into the data center management part of a DCI-OS and relates to following functionality:

- *Planning and Capacity Management* for Application Services – based on projected information, new Applications Services may need to be accommodated in a data center while others retire. Workloads must be predicted for Application Services such that appropriate types and quantities of data center components can be projected that will deliver needed capacity in future.
- *Inventory Planning* – is derived from capacity management and translates into decisions which data center components (e.g. servers) will be acquired and made available to meet future demands.
- *Inventory Management* – discovery and verification of presence and state of data center components.
- *Data Center Component Management* – to manage data center components. Since those components are procured in larger numbers of same type, and further streamlining of IT inventory towards more homogeneity supports this trend, data center components of same type can be used interchangeably and hence managed in *Pools*.
- *Pool Management* is in scope of a DCI-OS since it must be able to acquire and build new resources. Pool management includes tasks to add or remove components from pools and keep track of their states and current and future assignments. The concept of Pools can not only be applied to bare data center components, often needed Infrastructure Resources, such as virtual machines, can be pre-created and maintained in Pools from where they can be assigned quickly.

  Operations on Pools include:

  - *Add and Remove Members* – of the same type increasing and decreasing pool capacity.
  - *Allocation* – maintaining information about planned future use of pool members. Pool capacity can be reserved to accommodate future needs by Application Services. Allocation includes operations to release, change or withdraw allocations.
  - *Assignment* –fulfillment of an allocation with any instance assignment of pool members at due time. Assignment includes operations to release, change or withdraw assignments.

Most functions from the above list can be provided in form of Infrastructure Management Services operating in the management environment.

3.5.3 Concept Generalization in the Organizational Dimension

Organizational concepts refer to the overall organization of a system, in particular how it relates to its environment and which information the system can rely on. Examples of organizational concepts are time-horizons and defining behavior such as by setting policy the system should follow.

Table 3 shows organizational concepts which have successfully been applied in operating systems.

Concept	Description
Comprehensive View of the Environment (Information Model – IM)	In contrast to operating systems, which have a comprehensive and exclusive view of their environment (the surrounding layers), IT management systems have only a limited view into their environment of a data center, typically related to the specific function there are aiming at. Network management systems, for instance, know little or nothing about storage or applications. The *Information Model (IM)* of a DCI-OS is at the core of its ability to automate IT management tasks. The Information Model must accommodate a comprehensive view of the environment, specifically information about layers that are associated with a DCI-OS (see Figure 2): • The Application Services and the Application Instances from the Application Environment, • The Application Execution Services and Infrastructure Services from the Application Execution Environment, with Topologies of Infrastructure Resources, organized as Pools, mapped onto • Data Center Components.
Scope in time of the Information Model	An Information Model maintains information about a defined entity domain over a certain time horizon. In contrast to an operating system, which mainly maintains timely and short-term information, a DCI-OS must cover a broader timeframe extending into the past and future: • Extending modeled information into the past – refers to keeping information about past states in the system. Purpose is to use this information for decision making purposes (e.g. trending) and for analysis (e.g. root cause analysis). Monitoring has always been a core part of IT Management for these reasons. A DCI-OS must be made aware of monitoring data that is collected in an environment (this means that DCI-OS itself does not need to implement monitoring, but it must connect to external monitoring Infrastructure Management Services. One extension over existing monitoring systems is that the DCI-OS must take control over the entities and relationships about

	which monitoring data is collected. Particularly the relationships must be captured flexibly in a dynamic environment where relationships may change over time, e.g. the assignment relationship of an Application Instance to a server resource. • Capturing present information – primarily refers to accurately and consistently capturing the actual states and situation in the managed environment. In IT management, this is a serious problem to capture accurate state of the environment. Reasons are the time-delay during poll cycles, the chance to not report events and the change in the environment itself. Not just data center inventory and components may change, in a more dynamic environment; also the relationships to resources and applications can change, sometimes frequently. One approach to overcome some of the issues is that the traditional ways to establish an Information Model in management systems primarily rely on information *discovered* in the managed environment and changes occurring in the managed environment must be propagated back into the Information Model. What has been missing is the fact that most changes are planned changes, which are designed, approved and then implemented in the managed environment. If this information were available in the Information Base of a DCI-OS a priori to a change (e.g. as a design or a planned change), then the change to the environment could not just be automated and driven out into the environment from the Information Model, it would also be known what is *supposed to be there in the managed environment* and verified whether this is the case rather than periodically scanning for changes. The Information Model of a DCI-OS hence should incorporate designs and changes to occur, which will lead to a notion of Desired State and Observed State discussed later. • Extending modeled information into the future – directly relates to the point to not only represent information about past and present states about the managed environment, but also future states. Future states of the managed environment can be represented in form of - Designs, based on which new future resource and service configurations can be produced by the DCI-OS, - Future schedules, based on which allocation decisions can be made in Pool management, - Planned changes to existing resource and service configurations. The information related to these aspects today is primarily known and managed by people in IT. An Information Model of a DCI-OS must incorporate particularly information about future states in order to:

	- drive automated creation, e.g. the generation and application of configurations in order to produce the creation of resources automatically as opposed to manually (with Topology Designs as foundation of this automation); - decide about resource allocation and assignment automatically, which assumes information about future resource needs; - guide capacity planning of Resource Pools; - apply changes automatically to the managed environment. *Automation* of provisioning and deployment processes fundamentally relies on information being available about future and current states of the managed environment.
Scope in space of the Information Model	An Information Model of a DCI-OS must incorporate information about elements of two main domains: • *Infrastructure Services* and *Infrastructure Resources* presented into the context of an Application Execution Environment (the application view); and • *Resource Pools* and *Data Center Components* (the data center view). The domain of Infrastructure Services and Infrastructure Resources is associated with concepts of: • *Topology* – describing a set of relationships among Infrastructure Services and Infrastructure Resources that are needed at once by an Application Execution Environment (e.g., a three-tier server environment with storage and network connections for a web application); • *Resource Construction Relationships* – describing the mappings of Infrastructure Resource onto underlying Infrastructure Resources or Data Center Components. Mapping relationships can cross application domains when applications share common resources, such as applications co-located on the same physical server. Complex Resource Topology and Resource Construction Relationships may occur, which need to be designed properly using design tools. Design tools produce machine-readable models of IT configurations. Blue prints of those designs can be created and reused for generally applicable designs, such as an IT design for a database server, which is an often found building block in enterprise applications. The use of design tools is critical for constructing complex resource topologies and resource construction relationships and also for maintaining them later when the system us in operation and change must be incorporated. Changes then first can be discussed and evaluated against designs before implementing them in the system.

Policy	Policy is understood as a mean to express a goal an automated system should follow for making its decisions. Policy consequently is essential for automation systems. A DCI-OS needs to incorporate a number of policies for making decisions about automated: - Resource Creation and Transformation, - Resource Allocation and Assignment, - How to achieve a desired state of a Resource Topology.
Sharing	The concept of Sharing in a data center environment occurs in dimensions of: - Sharing resources among applications for purposes of communication or collaboration. - Multiple application environments sharing a set of components in a data center.
Isolation	The concept of Isolation in a data center environment occurs in dimensions of: - Avoid unwanted direct interference between application environments. Operating systems implement this concept by isolating address spaces and controlling visibility and access to resources associated with applications. - Allow control over unwanted indirect interference between applications that results from the shared environment in which they exist. In particular performance effects may result from interference in the underlying shared environment.

Table 6: Generalized IT Management concepts in the organizational dimension.

The concepts presented in Table 1 to Table 3 represent fundamental concepts which have been adopted from operating systems and reinterpreted for the context of data center management. These concepts will be used in the following chapter to outline in more details the structural, functional and organizational requirements for a DCI-OS.

Chapter 4

Requirements Analysis for the DCI-OS

The discussion in this chapter will identify requirements for a DCI-OS beginning with an analysis of an early integrated data center management and automation product which was called the HP Utility Data Center (UDC) [UDC]. The UDC came to market in 2002 and was discontinued in 2004 as a failure. He failure was caused by a number of market reasons such as high upfront cost, but was also caused by a number of technical shortcomings which were analyzed to provide the basis for a better data center automation product with the overall approach to research not only the shortcomings, but also take knowledge into account from related disciplines of operating systems and data center management.

This chapter first analyses the structural, functional and organizational shortcomings of the early UDC data center automation product in section 4.1 to then establish a set of structural, functional and organizational requirements for a DCI-OS taking ideas and patterns from operating systems and data center management into account in sections 4.2, 4.3, and 4.4. These general requirements are then translated into a specific set of technical requirements, which are presented at the end of this chapter in section 4.5.

These requirements form the basis for defining the architecture of the DCI-OS in the following chapter.

4.1 The Starting Point for Requirement Analysis

The value proposition for the UDC solution was to increase automation of data center management tasks leading to significantly lower operational cost. However, while automation of lower level operational management tasks could be achieved, the overall results were not satisfactory. Furthermore, the overall cost of the UDC solution was exceeding the anticipated cost savings significantly. With the ambitious cost savings not being realizable and high upfront cost, the UDC failed at the market as a commercial offering and was withdrawn in 2004.

It was however recognized that increasing automation in data center management should be continued, leading to the research effort this thesis is based on.

The UDC also suffered a number of structural, functional and organizational shortcomings, which are discussed later in this section. Detailed analysis of these shortcomings then provided the basis for the requirements for future developments.

Figure 4: Control racks of the Utility Data Center (HP Labs, Palo Alto, Nov 2004).

Figure 4 shows the installation of the control racks in the HP Labs Palo Alto Data Center in November 2004. These units constituted the control systems of the UDC, not the actual managed systems. Due to the nature of centralized control of the entire(!) data center through one control unit, this units had to be highly reliant. Failure of the control unit would have a data center-wide impact and hence had to be avoided.

Substantial engineering effort was spent to build out the control unit with three replicated instances, which had to synchronize their states and be able to accept the role of a master controller at any time.

Figure 4 shows the storage arrays with three cabinets on the left. Two central SAN networking switches are shown in the center. And to the right, there are three racks with a number of server blades running the controller software. The right-most rack also has the operator's console showing the design of a resource configuration, which will be explained later in more detail. The abstraction of a so-called ***Resource Topology*** was an essential innovation the UDC solution incorporated, which was further developed later on.

Another key innovation was the concept of a ***Design***, or a computer-represented model, of a Resource Topology, which also was adopted and further developed in future research.

Both concepts also represent examples of concepts which did not exist in operating systems and hence needed to be developed for the context of a data center.

The technical foundation of UDC's capabilities was based on the assumption that modern data center resources are all programmable, including connection networks (LAN, SAN), which means they can be accessed not only via management consoles by operators (people), but also through API. By programming, "virtual connections" could be established between any resource pair.

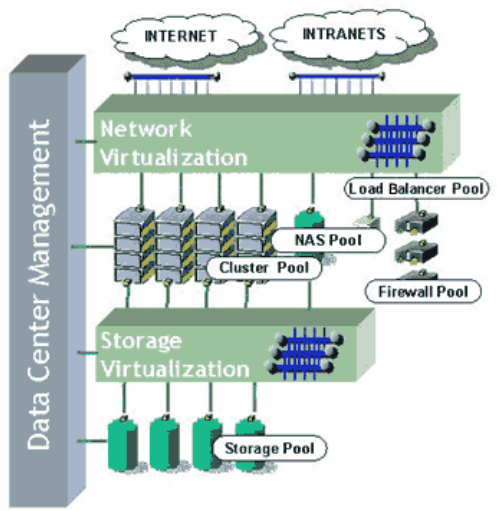


Figure 5: Programmable resources as basis for the Utility Data Center (UDC).

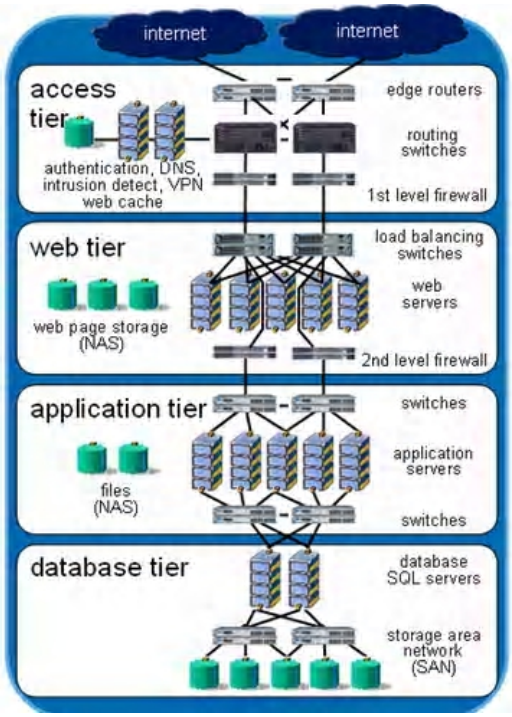


Figure 6: Resource Farm as abstraction for a configured resource set.

Figure 5 shows the two control planes of the UDC with programmable network fabrics for connecting machines to storage (via the programmable SAN) and for connecting machines to other machines and devices (via the programmable LAN). The control layer of the UDC is shown on the left hand side which allowed the coordinated creation of the proper connections among sets of resources as desired.

The control logic allowed creating and executing task procedures which can be invoked and, internally, breaking tasks down into finer grain automation procedures affecting individual resource sets and connection networks. This procedural programming capability allowed automation to a much higher degree than it was possible before.

The UDC solution provided an abstraction for configured resource sets called *Farm*. A Farm included a number of resources configured in a way such that they formed an execution environment for structured applications which could be deployed into this resource environment later.

Figure 6 shows three Farms containing different resource sets tailored for three-tier applications, which had become popular in data centers. Those applications consisted of an access tier with edge routers, routing switches and firewalls providing authentication, DNS, intrusion detection, VPN capabilities and caching; a web tier handling incoming HTTP requests with load balancers and a scalable set of web servers; an application tier with another scalable set of servers running application servers and logic; and the final database tier where all persistent state was stored requiring server resources for the database and storage resources for disks.

A simple description language was defined for the UDC which allowed describing the structure (~topology) of the resource sets as well as the automation procedures. The language was XML-based and called Farm Markup Language (FML) [FML].

Farm descriptions in FML could be submitted to the UDC control software for creating the described farm. The UDC control software was able to create the described resource topology by invoking automation procedures it had built in.

Despite the fact that the UDC at the time was the most advanced integrated data center automation product, the UDC suffered a number of shortcomings which contributed to its failure in the market place.

These shortcomings were analyzed and formed the basis for defining the requirements for the DCI-OS aiming at overcoming UDC's shortcomings. These requirements are presented in detail in sections 4.2, 4.3, and 4.4.

Three dimensions have been identified for the shortcomings of the UDC:

- structural shortcomings,
- functional shortcomings, and
- organizational shortcomings.

Structural shortcomings

Structural shortcomings primarily relate to the structure of UDC's control software and internal data model. The control software was a monolithic block of tightly coupled software components. Internal software blocks used specific functions of vendor-specific hardware capabilities making it hard, and later impossible, to keep software blocks up to date with fast replacement cycles of the hardware. The set of supported hardware lacked by two generations behind the latest hardware components because of the need to update internal software components.

Breaking the monolithic structure apart and introducing structural concepts known form operating systems such as Hardware Abstraction Layers (HAL) and a Device-driver architecture which allows hiding changes in underlying hardware components, or at least keeping them local in device derivers, has been seen as an approach overcoming the structural shortcoming of the monolithic structure of UDC's control software.

In terms of the data model, the UDC used a standard SQL Schema of a fixed set of entities and relationships to represent the resource sets of a farm and other entities it needed to maintain internally. The disadvantage of this approach has been that the SQL model lead to the interlock between the data structures defined in the schema and the code that operated upon those data structures. When the schema was changed, the code had to be changed as well with no explicit isolation into modules which would be defined for certain parts of the schema. Code/data isolation was not considered and hence leading to potentially global changes in the code base as new functions or capabilities or advancing hardware needed to be incorporated. Although the data model used the abstractions provided by a SQL database, its entity representations were proprietary and not extensible (because of the data/code lock-in).

Functional shortcomings

Functional shortcomings relate to the functionality the UDC provides as a system, which related to basic hardware components, but did not consider deployment and configuration

of software. For example, the UDC could configure the server with disks and network interfaces for a database application, but could not provision the software onto the server, including the operating system. These tasks still required human action and limited the automation capabilities substantially.

Another functional shortcoming was that configurations were considered statically by the UDC. This meant that, once made, they could not be changed unless the overall farm was dissolved. There was no dynamic resource management, resource provisioning on demand of resource flexing capabilities, which again posed substantial limitations to customers' expectations. The UDC also did not manage resources in a sense that it kept track of which resources were assigned to which farm deployments. Resource allocations and resource assignments were static and had to be organized by human operators requiring the explicit specification of resource instance identifiers in FML specifications.

One effect of this circumstance was that FML specifications were not reusable because they contained physical resource identifiers, such as Ethernet addresses for servers or Logical Unit Numbers (LUNs) for virtual disk from storage arrays, which had to exist and created manually before the Farm could be deployed. These examples demonstrate how small flaws in the architecture of the data model lead to significant functional limitations in the final product.

Organizational shortcomings

Organizational shortcomings relate to the way the UDC organized the environment of resources under its control. The UDC kept a current inventory of the current set of resources that existed in a data center. Changes to this resource set, e.g. when defective parts had to be replaced, had to be entered and updated manually via the console, which turned out to be major source of human-induced error. Automated resource discovery was added in later version of the UDC. The UDC, like most IT management systems, only maintained inventory information about managed elements which really existed in the data center as physical devices, such as servers, storage arrays, network switches and routers and firewall or load balancing devices. UDC's data model could not comprehend future inventory, which was not present in the data center yet, but was planned and known to exist at some future time, as well as inventory which existed at the current time, but was known to not exist in future due to replacement, for example.

The limitation in the data model to only consider current, physical inventory prevented automation of resource capacity management, future application deployments, coordination of future deployment plans, identifying resource conflicts as well as automated provisioning schemes (the UDC would select resources as opposed to rely on specified resources in FML). Resource allocation and assignment plans had to be maintained by operators manually and separately using Excel spreadsheets from where the correct information about the proper set of resources had to be copied over into the proper fields of the FML description submitted to the UDC, another major source of human-introduced failure.

These limitations, in many regards, could be overcome technically when they would be identified first, then represented as structural, functional and organizational requirements and finally translated into specific technical requirements to the structure, function and organization of what is called the DCI-OS in this thesis.

4.2 Structural Requirements for the DCI-OS

Ideas and patterns from operating systems and IT data center management should be taken into account for formulating the requirements for the DCI-OS. Along the structural dimension, the following concepts have been identified in section 3.5.1:

- *Layers* with Sub-layers and Modules,
- *Interfaces* between the Layers and Modules.

In addition, two stacks of corresponding layers have been identified for data center management:

- the *Managed Environment* and
- the *Management Environment*.

As specific Layers structuring the Managed Side have been identified:

- Application Layer with concepts of an
  - *Application Services Environment* with *Application Instances* forming and *Application Service*, and
  - *Application Services Execution Environment* with *Infrastructure Services* and *Infrastructure Resources* produced and delivered to Application Instances.

As specific Layers structuring the Management Side have been identified:

- *IT Infrastructure Service Management Layer* with concepts of *Infrastructure Management Services*. Further structure is provided as internal layers of:
  - *Infrastructure Management Services* – providing the actual management functionality,
  - *Hardware Component Abstraction Layer (HCAL)* – providing a uniform access layer to data center components for higher-ordered Resource Management,
  - *Hardware Component Drivers* – for accessing data center components via their control interfaces and the
  - *Data Center Component Management Layer* with concepts of *Component Interfaces* to data actual physical components found in a data center.

Conclusions for structural requirements for a DCI-OS are:

- Functional building blocks in the IT Infrastructure Service Management Layer are implemented as Infrastructure Management Services, which are application-level services operating on machines that are dedicated to management. Examples of such services are (they are discussed in more detail in the next chapter):
  - Resource Acquisition Management Service,
  - Provisioning and Deployment Service,
  - Task Automation Controller Service.
- Interfaces are implemented through standard middleware protocols allowing the interactions with the infrastructure management services.
- A modular structure supporting reuse and extensibility is another structural requirement for the architecture of a DCI-OS.

4.3 Functional Requirements for the DCI-OS

The following concepts have been identified along the functional dimension in section 3.5.2:

- The main function of a DCI-OS is to *provide the Application Service Execution Environment* with all needed Infrastructure Services and Infrastructure Resources for Application Instances and Application Services.
- A core function of a DCI-OS is the production of Infrastructure Services and Infrastructure Resources for Application Services.
- *Management Tasks* also need to support the concept of an Application Service in form of its Application Service Lifecycle Management with tasks of:
    - Application Service Design with Grounding and Development of IT Design and Topology Models,
    - Resource Acquisition and Provisioning,
    - Deployment, and
    - Operational Management.
- *Task Automation* for management functionality is an important functional goal of a DCI-OS reducing the need for human action or intervention when possible.
- *Protection of resources* with regard to access control and isolation must be supported by a DCI-OS in order to avoid unwanted interferences among Application Services which coexist in a shared data center environment.
- *Production of Infrastructure Resources* is an essential function of a DCI-OS which is achieved by applying configurations to data center components or recursively to underlying previously produced Infrastructure Resources making them properly configured and usable for an Application Service.
- The following categories of resource productions referred to as *Resource Transformation* are distinguished:
    - *Resource Generation* – the creation of a new resource type as composition of underlying resources in combination with specific configurations; an example is a general purpose file server resource which combines a server resource, storage resources and file server operating software;
    - *Resource Virtualization* – the multiplication of underlying resource types with same properties; an example are virtual networks, virtual disks or virtual machines; and
    - *Resource Creation* – the production of a specific resource type with application-specific properties fitting the particular needs of application services; an example is a server configured with a database that can be seen as a database resource for a particular application service.
- *Scheduling* is the process of coordinating resources that are shared among Application Services. Likewise in operating systems where resources are shared among application processes, scheduling is a concept that is important in the context of data center resources as well. Scheduling includes resource requirement use and

conflict detection and resolution and the creation of a resource allocation and use plan. Scheduling is an important function of a DCI-OS.

- *Accounting and Controlling Resource Use* is another function required for a DCI-OS including monitoring and associating monitored data with the contexts of Application Services. Control needs to be applied when resources used by Application Services exceed their previously established limits.

These functional requirements can be sees as common between operating systems and the data center. The differences are in the kind of resources that are managed in a machine versus in a data center. But the overall functionality is common.

There is some substantial functionality that is missing or only present in rudimentary form in operating systems that must be supplemented in addition to complete the functional requirements for a DCI-OS.

This missing functionality primarily relates to aspects of:

- *Data Center Planning, Design and Management* with support of:
    - *Planning and Capacity Management for Application Services* - based on projected use information; new Applications Services may need to be accommodated in a data center while others retire. Workloads must be predicted for Application Services such that appropriate types and quantities of resources can be produced in current or future data center components such that the expected (known) Application Services can be accommodated.
    - *Inventory Planning* - is derived from capacity management and translates into decisions which data center components (e.g. servers) will be acquired and made available to meet future demands.
    - *Facility Planning* – as the processes needed to provide the physical environment in which current and future data center components can exist.
    - *Regulatory Planning* – as the processes needed to meet all regulatory obligations.
- *Inventory Management* - discovery and verification of the presence and state of current data center components. In more advanced systems, also future inventory is managed which is known to exist at some time in future as well as other components which are known to not exist beyond a certain time in the future.
- *Data Center Component Management* – addresses the management of data center components. Since those components are typically procured in larger numbers of same type, and further streamlining of IT inventory towards more homogeneity supports this trend, data center components of same type can be used and managed more uniformly and interchangeably in *Pools* as opposed to individual instances.
- *Pool Management* enables the acquisition of resources of the same type. Pool management includes tasks to add or remove resources from pools as well as keeping track of their states and current and future assignments. The concept of Pools can not only be applied to bare data center components, it can be applied to constructed resources as well, such as virtual machines which are pre-created and also maintained in Pools from where they can be assigned for use quickly.

Operations on Pools include:

- *Add and Remove Members* - of the same type increasing and decreasing pool capacity.
- *Allocation* - maintaining information about planned future use of pool members. Pool capacity can be reserved to accommodate future needs by Application Services. Allocation includes operations to release, change or withdraw allocations.
- *Assignment* - fulfillment of an allocation with any instance assignment of pool members at due time. Assignment includes operations to release, change or withdraw assignments.

Most functions from the list can be provided in form of Infrastructure Management Services operating in the management environment on the data center side of a DCI-OS.

Later discussion in this chapter will use a numbering scheme for a more refined list of requirements. This numbering scheme uses a letter *R* followed by a three-digit number *R.ddd* where the first digit represents a class of requirements. This requirement numbering scheme has been established in HP and is used here as well.

The following list shows the relationships between the more general functional requirements discussed in this section and how they relate to specific functional and organizational requirements against which the architecture then must be evaluated.

Using the *R.ddd* scheme, the refinements of functional requirements relating to the Data Center side and the side of Infrastructure Services can be categorized into:

- Refinement of functional requirements for a DCI-OS related to Resource Management in a data center (R.100):
  - Resource capacity and inventory planning (at future time, R.110),
  - Resource capacity and inventory management (at current time, R.115),
  - Resource categories (basic, transformed, R.120),
  - Organization of resources (pools, R.140),
  - Resource sharing (R.150),
  - Resource acquisition from resource pools (allocation, assignment, R.160),
  - Automated resource pool management (add, remove, replace resources,R.170),
  - Resource programmability (apply configurations programmatically, R.190),
  - Resource pool monitoring and accounting (monitor, account, R.200),
  - Discover, probe and report resource inventory (R.210).
- Refinement of functional requirements for a DCI-OS related to Infrastructure Service management (R.300):
  - Topology design and models (R.300),
  - Automated resource acquisition (allocation, assignment, release, flex, R.400),
  - Automated deployment (resources, applications, R.500; deployment control R.600),
  - Automated operational management (set policy R.700; control lifecycle R.800).

4.4 Organizational Requirements for the DCI-OS

As organizational concepts have been identified:

- *Comprehensive view of the environment through Information Model* – The Information Model must accommodate a comprehensive view of the environment, specifically information about layers that are associated with a DCI-OS (see Figure 1): The Application Services and the Application Instances from the Application Environment, The Application Execution Services and Infrastructure Services from the Application Execution Environment, with Topologies of Infrastructure Resources, organized as Pools, mapped onto Data Center Components.
- *Scope in time of the Information Model* – Extending modeled information into the *past* - refers to keeping information about past states in the system. Purpose is to use this information for decision making purposes (e.g. trending) and for analysis (e.g. root cause analysis). Capturing *present* information - primarily refers to accurately and consistently capturing the actual states and situation in the managed environment. The Information Model of a DCI-OS should incorporate designs and changes to occur, which will lead to a notion of *Desired State* and *Observed State* of system elements. Extending modeled information into the *future* - directly relates to the point to not only represent information about past and present states about the managed environment, but also future states. Future states of the managed environment can be represented in form of Designs, based on which new future resource and service configurations can be produced.
- *Scope in space of the Information Model* – An Information Model of a DCI-OS must incorporate information about elements of two main domains: Infrastructure Services and Infrastructure Resources presented into the context of an Application Execution Environment (the application view); and Resource Pools and Data Center Components (the data center view).
- The domain of Infrastructure Services and Infrastructure Resources is associated with concepts of:
    - *Topology* - describing a set of relationships among Infrastructure Services and Infrastructure Resources that are needed at once by an Application Execution Environment (e.g., a three-tier server environment with storage and network connections for a web application);
    - *Resource Construction Relationships* - describing the mappings of Infrastructure Resource onto underlying Infrastructure Resources or Data Center Components. Mapping relationships can cross application domains when applications share common resources, such as applications co-located on the same physical server.
- *Policy* – allows expressing a goal an automated system should follow for making its decisions. Policy consequently is essential for automation systems. A DCI-OS needs to incorporate a number of policies for making decisions about automated: Resource Creation and Transformation, Resource Allocation and Assignment, How to achieve a desired state of a Resource Topology.

- *Sharing* – in a data center environment occurs in dimensions of: Sharing resources among applications for purposes of communication or collaboration. Multiple application environments sharing a set of components in a data center.
- *Isolation* – in a data center environment occurs in dimensions of: Avoid unwanted direct interference between application environments. Operating systems implement this concept by isolating address spaces and controlling visibility and access to resources associated with applications. Allow control over unwanted indirect interference between applications that results from the shared environment in which they exist. In particular performance effects may result from interference in the underlying shared environment.

4.5 Requirement Mapping and Refinement

Based on the discussion of structural, functional and organizational requirements in the previous sections, those requirements can now be mapped into technical properties based on which the architecture can be formulated later.

A duality exists between an environment of an operating system and a data center. There are the resources that exist in the environment that need to be managed, and there are the applications consuming resources, which also need to be managed. Both management domains have different purposes. Managing resources from an overall data center perspective relates to planning capacity, managing inventory and managing the overall data center operation with respect to its resources. Managing resources from an application perspective relates to the particular set of resources consumed by the application. In IT management, both domains are typically clearly separated: the data center management and application management.

This principal separation of concerns also exhibits in the mapping of requirements into technical properties. The following two sections discuss the requirements from the

- *perspective of the data center*, which is discussed in section 4.5.1 below, and the
- *perspective of individual Infrastructure Services*, which is discussed in section 4.5.2.

4.5.1 Refinement for the Data Center Perspective

The following principles guide the definition of requirements for a DCI-OS from the perspective of *Data Center Management*:

- Support for capacity management and data center planning.
- Ability to deal not only with current inventory, but also with projected future inventory.
- Manage capacity rather than inventory, particularly in the future.
- Single abstraction for data center resources: *Resource Pools*.
- Automated resource pool management.
- Support for transformed resources, specifically virtual resources in pools.
- Support for sharing resources.

- Exclusive access to data center components through programmatic management interfaces eliminating direct human access to data center components.
- Automated discovery and consolidation of current with projected inventory in the data center information model.

The following table presents the requirements for a DCI-OS for Data Center Management.

Requirements **Data Center Management**	**Definition**	**R.100**
Resource capacity and inventory planning	A DCI-OS must support planning processes of future resource build-outs of a data center.	**R.110**
- DC capacity planning (future)	Capacity management is critical to determine estimates of future resource capacity needed in a data center.	R.111
- DC inventory planning (future)	Inventory management must incorporate future resource capacity in order to support longer-term resource allocation. Inventory management affects changing resource quantity as well as changing resource types.	R.112
- Abstract capacity from inventory (future)	Properties of future resources may not be known. Consequently, future needed capacity must be kept independently from inventory.	R.113
Resource capacity and inventory management	A DCI-OS must support current management of data center capacity and inventory.	**R.115**
- DC capacity management (current)	Current data center capacity must be managed against current demand.	R.116
- DC inventory management (current)	Current data center inventory and its use must be managed.	R.117
- Discover inventory and compare with plan	Inventory management must probe the existence and state of inventory in the DC and compare it with the planned inventory.	R.118
- Expose DC inventory programmatically	In order for a requestor to discover which resources are available in a DC, its inventory (resource types, capacity, and availability) must be exposed programmatically.	R.119
Resource categories	A DCI-OS must support the variety of resources occurring in a data center.	**R.120**
- Resource types	Resources of same kind are classified as resource types.	R.121
- Expandable resource types	With progressing technology, new resource types (vendor provided) must be incorporated.	R.122

- Customizable resource types	Allow introduction of new resource types provided by data center operator.	R.123
- Resource instances	Allow introduction of new resource instances:	R.124
- replacements	as replacements or	R.125
- expansion	as expansion of existing resource capacity.	R.126
- Basic resource types	All basic resources in a data center must be supported. Basic resources are resources as found in a data center.	R.130
- servers	Servers of different types must be supported.	R.131
- storage	Storage of different types must be supported (built-in disks, arrays, SAN arrays, NAS).	R.132
- networks	Support for different network types (LAN, VLAN and SAN).	R.133
- devices	Devices such as load balancers, fire walls, appliances, routers, switches must be supported.	R.134
- Transformed resource types	Resources can result from transformation processes performed on underlying basic resources. It is common to transformed resource types that a relationship to underlying resource types must be maintained.	R.135
- generated resource types	Generated resources provide capabilities that are required by applications. Those capabilities are different from the capabilities provided by underlying basis resources. Examples are disks generated in a storage array or subnets generated in a general network environment. Generation of resources is actuated by applying configurations to underlying basic resources.	R.136
- types of virtual resources	Virtual resources are multiplied resources of underlying types with same capabilities. A virtual machine is an example or a resource partition. Allowing for resource sharing is often a reason for using virtual resources.	R.137
- Resource identity	Resource instances must have a unique identifier (unique in time and space, not reusable). This is particularly important for transformed (generated, virtualized) resources.	R.138
- Resource transformation relationships	Relationships of transformed resources to their underlying resources must be maintained.	R.139

Organization of resources	Resources of multiple, varying types with multiple, varying numbers of instances must be organized in a common abstraction.	**R.140**
- Resource pools	A resource pool contains resource instances of the same type.	R.141
- Time-varying pool capacity	Capacity of the resource pool (as number of instances) can vary over time.	R.142
- Integration with planning	Planned capacity and inventory from R.111, R.112 must be factored into future states of resource pools.	R.143
Resource sharing	Sharing resource must be supported by a DCI-OS, with or without awareness of applications.	**R.150**
- Explicit sharing	Explicit sharing means that applications are aware of sharing. An example is a multi-tenancy use of a database.	R.151
- Implicit sharing	Implicit sharing means that multiple resource assignments are mapped onto the same resource instance without awareness of applications. Virtualization technology is often used to isolate shared resources.	R.152
Resource acquisition from resource pools	Resource acquisition in a data center means making resources available for applications from resource pools.	**R.160**
- Resource allocation	A resource allocation is a commitment the resource pools provides to a requestor of a defined resource capacity of requested resource types for a defined use period. Resource allocations can be requested and as result refused or committed by a resource pool. Resource allocations can be withdrawn or change requested by the requestor.	R.161
- long-term, future	A resource allocation can be issued by the requestor at planning stages; neither the requesting application nor the resources must exist in a resource pool.	R.162
- short-term, current use	A resource allocation can be issued during run time of an application to adjust its future resource use (increase, decrease).	R.163
- Multi-type resource allocation	Applications will need sets of resource types at once on order to function leading to the requirement of multi-type resource allocations.	R.164
- Resource assignment	A resource assignment is a fulfillment of a resource allocation at the time it has become due. Free resources from current resource pools are selected and assigned to the requestor or requesting application.	R.165
- Multi-type resource assignment	Applications need sets of resource types at once. A DCI-OS must support multi-type resource assignments.	R.166

Automated resource pool management	Pools of resource types are constantly filled with resource instances as they become available in a data center.	**R.170**
- Add resources	New resources must be added to existing pools. If resources of new types arrive for which no pool exists, a new resource type along with a new pool must be created.	R.171
- Remove resources	Retired (e.g. end of life) or failed resource must be eliminated, eventually entire pools must be eliminated as well.	R.172
- planned resource replacement	Planned retirement of resources or failed, unassigned resources must be removed from pools and eventually replaced with other resources of that type to maintain current resource pool allocations.	R.173
- Resource pool drivers	Within a DCI-OS, resource pool drivers produce the resource types by applying configurations to basic resource components.	R.174
- Resource replacement due to failure	Unplanned failure causes the need for replacing resource instances.	R.180
- passive resource instances	Failed resources that have not been assigned must be removed from the pool and eventually be replaced.	R.181
- active resource instances	In case assigned resource instances fail, they must be replaced for the assignment with other compatible resource instances.	R.182
- programmable resource identity	In case a resource can shield it's identify from an application by allowing to program its identity, a different, compatible instance can be plugged into an assignment without causing change for the application.	R.183
Resource programmability	In order to automate resource pool management, resource configurations must be programmable to resource components.	**R.190**
- Wire once, programmable configurability	Programmability of resources also means that physical wiring must only be done once while any desired configuration can be applied programmatically.	R.191
- Remote access to resources	Programming resources from a DCI-OS assumes remote, programmable access to resource components directly or to components which make resources available.	R.192
Resource pool monitoring and accounting	During operation, resource pools and resources in pools must be monitored.	**R.200**
- Detection of failing or failed resources	Detection of failing or failed resources triggers R.181 or R.182.	R.201

- Monitor resource use	Resource use must be monitored during an assignment.	R.202
- Resource use logging and accounting	Resource use must be logged and accounted to a requestor's account.	R.203
- Monitoring of resource pools	Resource pool utilization must be monitored and compared with projections.	R.204
- Performance monitoring	If applicable, performance of resources must be monitored.	R.205
Discover, probe and report resource inventory	During operation, resource inventory must be discovered, its state probed and detected and reported to resource inventory management.	**R.210**
Detect and discover resource inventory	Discoverable data center components must be detected and reported to inventory management.	R.211
Probe state of resource inventory	State of inventory must be probed and reported to inventory management.	R.212

Table 7: DCI-OS requirements supporting data center management.

4.5.2 Refinement for the Infrastructure Service Perspective

The following principles guide the definition of requirements for a DCI-OS from the perspective of *Infrastructure Services*:

- Model-driven design and management based on a common abstraction: *Topology*.
- Modularity and reuse of topology designs.
- Openness and modularity of model layer.
- Multi-instantiation and multiplication of topology designs in a data center.
- Portability of a topology design into other data centers.
- Support for transformed resources, specifically virtual resources.
- Isolation between resource specifications and bindings to actual resource instances; late binding between resource allocations and resource assignments.
- Apply formal methods to ground and size a topology for requirements.
- Automated resource acquisition.
- Automated deployment (concept of an automation controller).
- Automated, policy-driven operational management including automated correction of certain error conditions.

The following table presents in detail the requirements for aspects Infrastructure Service management in a DCI-OS as the execution environment of an application service.

Requirements **Infrastructure Service**	**Definition**	**R.300**
Topology design and models	In order to automate the creation of infrastructure services, formal descriptions of the to-be infrastructure service must exist in form of formal models at planning and design stages.	**R.300**
Topology	A number of resources (basic and transformed) must be considered as a whole in order to provide the execution environment of a desired infrastructure service. All resources needed must be present, must be properly configured in associated states. The concept of a *topology* forms the basic abstraction for modeling (planning and designing), deploying and operating an infrastructure service.	**R.301**
- Topology design of infrastructure service	A *topology design* is a graph structure which describes elements (resources) and relationships supplemented with all necessary configuration data needed for creating an infrastructure service. A DCI-OS must be able to interpret a topology design in order to automatically create an infrastructure service. A topology design can also be seen as the definition of an *abstract machine* as execution environment for application services.	R.310
- Resource grounding	*Grounding* is the process of resolving higher-order resource requirement descriptions into concrete resource types and quantities available in a data center for request. For example, a server is needed in a resource topology design which can support a mySQL database. This requirement can be grounded into multiple server types, depending on availability of those types in a data center. In order to ground resources, a DCI-OS provides information about the data center resource inventory at deployment time.	R.311
- Resource sizing	*Sizing* is the process of determining capacity of those types such as number of instances (e.g. servers), transaction rate, bandwidth or storage capacity. Resource grounding and sizing leads to a more specific *topology design variant*.	R.312
- Multi-grounding support for topology design	Multiple choices can exist for grounding and sizing resource requirements in a topology design. Exploring multiple choices, as well as resulting topology design variants should be enabled.	R.313
- Automated grounding and sizing	A DCI-OS should support automated grounding and sizing.	R.320
- Reusability of topology designs	One significant advantage of using topology designs as models of to-be infrastructure services is reusability of those designs.	R.330

- re-instantiation	A topology design must be re-instantiatable independently how many times it has been instantiated before.	R.331
- multi-instantiation	A topology design must be instantiatable multiple times in a data center leading to multiple instances of infrastructure services.	R.332
- Portability and reuse of topology designs	A topology design should be portable such that it can be reused without or with little modification.	R.340
- across changes in a data center	A topology design should be portable across changes occurring in a data center, such as when resource inventory is updated.	R.341
- across data centers	A topology design should be portable across data center tolerating the changed environment as much as possible.	R.342
- Customization of topology design	Changes in requirements in a topology design can lead to different results in grounding and sizing supporting topology design variants. Based on the initial topology design with the highest-level resource specification, customizations should be derivable based on which then supporting topology design variants should be generated.	R.350
- Operational management policies in topology designs	In order to drive automated operational management after creation of infrastructure service, operational management policies must be specified in the associated topology design. A DCI-OS must interpret those policies guiding its automated operational management.	R.360
- Desired State Model	Operational management policy is defined as an adjustable *Desired State Model* of the infrastructure service against which a DCI-OS will evaluate the observed state of the running infrastructure service.	R.361
- Observed State Model	The Observed State Model (at design time) defines the elements, states and transitions that need to be observed later when the infrastructure service is operating.	R.362
Automated Resource Acquisition	Automated resource acquisition includes all stages related to acquiring, using and releasing the necessary set of resources for a resource topology. Resources acquisition follows a lifecycle of resources being *requested*, *allocated*, *assigned*, *used (owned)* and *released*.	**R.400**
- Resource allocation request	A request for resources must be issued to the DCI-OS containing the resource types and their quantities (quantity in terms of numbers, size or capacity) that are anticipated for use by a resource topology over a projected time profile.	R.410

	A *resource profile* is a multi-dimensional projection (one projection per resource type) of quantities over time. A request contains one profile and is issued for an *entire resource topology* with all its needed resource types and quantities. The DCI-OS responds with a grant or a denial of the request.	
- Resource request granted	If a resource allocation request is granted, *all* its components are committed by the DCI-OS.	R.411
- Resource request denied	If a resource allocation request is denied, no changes occur in the state of the DCI-OS.	R.412
- Resource allocation	A *resource allocation* is the commitment of the DCI-OS to a *requestor* of the availability of quantities (numbers, size or capacity) of resource types described in a granted resource request. If a resource request is granted, the projected resource quantities of the request are subtracted from the projected capacities of resource types in the data center.	R.420
- Resource allocation change request	Any granted resource request can be requested to change by the requestor, a third-party role or the DCI-OS. The DCI-OS responds by either granting or rejecting the change. A change request may refer to resource types, quantities or time profiles of projected use. A reduction in resource types or quantities is always granted by the DCI-OS. A change request is either fully granted or rejected.	R.421
- Resource allocation request cancellation	Any granted resource request can be requested to be cancelled by the requestor, a third-party role or the DCI-OS. Cancellation requests are always granted by the DCI-OS.	R.422
- Resource assignment	*Resource assignment* is the process of selecting individual resource instances from pools in order to fulfill a resource allocation at the time when it has become due, which means the first time of use as described in the request has arrived. The requestor becomes the resource *owner* with an assignment.	R.430
- Resource creation with assignment	Resources may be created by the DCI-OS in order to be assignable. Examples are virtual resources that are assigned.	R.431
- Initial deployment by DCI-OS	Before resources are handed over to the owner, the DCI-OS applies basic configurations to bare resources as specified in the resource topology. Requirement class R.500 describes automated deployment carried out by the DCI-OS.	R.432
- Resource hand-over	After assignment, resource instances are handed over to the owner of resources for subsequent deployment (configuration).	R.433

- Resource access by owner	The resource owner is notified by the DCI-OS of the assignment. Resource specific *handles* are passed with the notification through which the owner can access resources (e.g. IP addresses of servers or virtual machines).	R.434
- Owner-initiated release of resource instances	Resource instances in use can be released by the owner. The owner specifies resources to be released by the handles obtained from the DCI-OS after assignment. The DCI-OS may destroy released resources without preserving state. Resource handles invalid after release. Resources must be re-acquired through a resource allocation request (R.410), which may be denied. New resource instances with new handles and in initial state are returned.	R.440
- DCI-OS expresses desire to releasing resources	The DCI-OS may issue requests to owners of resources to release resources voluntarily. Owners may or may not respond to this request.	R.441
- Resource preemption	The DCI-OS may preemptively withdraw resources in order to fulfill higher prioritized requests. Potential resource preemption is announced by the DCI-OS to the owner before the preemption can occur. (controversial case).	R.442
Automated deployment	*Deployment* is the process of applying configurations to resources. Access to resources and eventual control interfaces must be given. Deployment is a multi-staged process. It begins with deploying configurations to bare resources as returned from the DCI-OS. Deployment occurs *before* resources are handed over to the owner of resources by the DCI-OS. Configurations are either known to the DCI-OS or are derived from the resource topology provided at deployment time.	**R.500**
- Basic resource deployment	Bare resource configurations are applied to bare resources. Main tasks are to establish the *identity* of the resource and the *interfaces* and *handles* handed over to the owner through which the resource is accessed later.	R.510
- Server deployment	Physical server deployment includes the creation of a server identity and network addresses.	R.520
- Server partition deployment	If applicable, a partition must be created on the selected server. An identity must be obtained or created for the partition.	R.521
- Virtual machine deployment	If applicable, a virtual machine must be deployed, created and started on the selected server. An identity must be obtained or created for the virtual machine as well as its network access.	R.522

- Storage deployment	Storage deployment depends on the kind of storage used. Following cases may occur.	R.530
- local disk deployment	If applicable, local disks need to be initialized.	R.531
- RAID deployment	If applicable, a RAID array must be configured.	R.532
- Array deployment	If applicable, virtual disks must be created in an array, LUNs must be obtained.	R.533
- SAN deployment	If applicable, LUNs attached to SAN cards in servers.	R.534
- NAS deployment	If applicable, file systems must be prepared in the NAS server for later server mount. LAN must be configured for NAS.	R.535
- Device deployment	Configure devices.	R.540
- Firewall deployment	If applicable, a hardware firewall is configured. An identifier must be obtained and its network access. The default firewall configuration is applied first. Any configuration obtained from the resource topology is applied after that.	R.541
- Load balancer deployment	If applicable, a hardware load balancer is configured. An identifier must be obtained and its network access. The default load balancer configuration is applied first. Any configuration obtained from the resource topology is applied after that.	R.542
- Network deployment	Network deployment achieves the creation of a functional network by applying configurations to routers, switches and other networking components.	R.550
- LAN deployment	LAN deployment creates IP subnets with routable IP addresses. LAN deployment affects either static IP addresses or DHCP.	R.551
- DNS deployment	DNS deployment creates routable DNS names in a LAN by configuring a DNS server.	R.552
- WAN deployment	WAN deployment creates access points to external networks. It can include gateways, proxies, firewalls, DMZ.	R.553
- SAN deployment	SAN deployment creates a functional storage area network between storage arrays and SAN access cards in servers.	R.554
- Software deployment	Software deployment makes the stack of software available on a server that is needed for its operation. Four major software layers are addressed: operating system, application software, application data and application customizations.	R.560

- OS deployment	Operating system deployment means to make a bootable operating system available on a server.	R.561
- OS installation Local disk ILO	An operating system is installed on a local disk of a server trough ILO (HP proprietary).	R.562
- Network installation	An operating system is installed on a local disk of a server through network installation such as PXE.	R.563
- SAN disk OS install	An operating system image is copied into a fresh SAN disk from a golden image.	R.564
- VM OS install	An operating system image is copied into a VM image.	R.565
- Application software deployment	Application software is installed on a disk that is made available to a server.	R.570
- Application data deployment	Application data is installed on a disk that is made available to a server.	R.580
- Customization deployment	An application software customization is installed on a disk that is made available to a server.	R.590
- Deployment control	Deployment control enables control over deployment processes.	R.600
- Start deployment process	A deployment process is started.	R.601
- Deployment state detection	State of a deployment process must be detectable: *not existing, created, starting, started, active, error, not responding, aborting,* and *aborted.*	R.602
- Terminate deployment process	A deployment process is aborted and a termination process is started which will abort a deployment process.	R.603
- Reset deployment state	Deployment state must be reset that may have been created as effect of a previous deployment process that has been aborted.	R.604
Automated operational management	Operational management enacts control over a managed environment. It is based on comparison of two distinct states: the desired state, which describes a state in which the managed system is expected to be in, and an observed state, which is a state in which the managed system currently is observed. An operational management controller aims to align the observed state to wards the desired state by issuing control actions to the managed environment, or if this is not possible, issue an alert signal to correct the situation.	**R.700**

- set Desired State	Operational control from an operator's point of view is achieved by defining the desired state.	R.710
- change Desired State	Desired state can be changed any time. Changes in desired state will likely trigger action in the associated managed system.	R.720
- Lifecycle control of a topology	Lifecycle control is a main aspect of operational control. It allows to start and to stop a system. Lifecycle control affects all elements of a topology. Its sub-aspects refer to individual components as well as to the topology as a whole. Lifecycle control begins after deployment and ends after termination.	R.800
- ignition	Ignition is the transition from an undefined state into a defined initial state. An example is bootstrapping a server. Ignition always starts with an ignition signal issued by an initiating device. Various ignition technologies exist such as PXE for network server ignition.	R.810
- start	Start is the transition from a defined initial state to an operational state.	R.820
- state detection	State of a topology as well as of individual components must be detectable any time.	R.830
- termination	Termination is the transition from a defined or undefined operational or error state into a defined terminated state, in which state of the topology still is in the system. Termination is the counter operation to start.	R.840
- destruction	Destruction is the transition from a terminated state where state is still present in the system to a non-existing state where no state of a topology is maintained in the system. Destruction is the counter operation to deployment of a topology. Destruction does not remove model data maintained about a topology.	R.850

Table 8: DCI-OS requirements supporting the management of infrastructure services.

The sets of requirements established in Table 7 and Table 8 present a comprehensive set of capabilities for a data center automation solution which go far beyond the capabilities the early UDC could achieve. Some of the requirements required deeper research, particularly for the planning and design phases, because these areas have not been covered by existing operating systems or IT management systems.

The following chapter presents the architecture of a DCI-OS based on these requirements. Concepts used in the architecture refer back to structural, functional and organizational concepts found in operating systems and reinterpreted for the context of a data center.

Major building blocks of the DCI-OS are presented followed by a discussion of how they support the requirements. A number of innovative properties result from the combination for a DCI-OS which are discussed at the end of the following chapter.

Chapter 5

Architecture of the DCI-OS

Figure 7 displays the architecture of a Data Center Infrastructure Operating System. The architecture presents two columns: a data center management-related part (on the left) and a part that is related to the management of infrastructure services (on the right).

An *Infrastructure Service (IS)* is a fully operational execution environment for an application service. It includes all necessary resources (servers, storage, and networks) in an operational state with proper configurations as well as all necessary application software and data made available and operational on those resources. An Infrastructure Service is the basic abstraction produced by a DCI-OS. It is comparable to a execution environment for a fully operational process created by an operating system in a computing machine

Multiple infrastructure services typically coexist in a data center supporting multiple application services. Consequently, the *Data Center (DC)* part is shared amongst all infrastructure services of this data center. The figure displays the shared data center part on the left and one infrastructure service as a representative on the right. There are connections between both parts through which information is exchanged and control is mitigated.

The vertical structure in Figure 7 in a DC and IS-related column supports reuse, portability and multiplicity of infrastructure services in a data centers. It is complemented with a horizontal structure in form of layers which reflect a further breakdown in each part into tasks and components.

Components in layers are further differentiated by different colors[1] in Figure 7.

[1] Green color in the architecture diagram in Figure 7represents operational technical modules that are part of the DCI-OS. Red color represents the information (models) needed to drive the technical modules. Brown color represents physical infrastructure components that are part if the data center infrastructure and external to the DCI-OS such as servers, storage and network devices. They are, however, subject to configuration and control from the DCI-OS modules.

The architecture consists of five layers. Adopting the concept of layers as architectural pattern is a first reference to modern architectures of operating systems, which also use layers to achieve abstraction, separation of concerns, modularization and reusability.

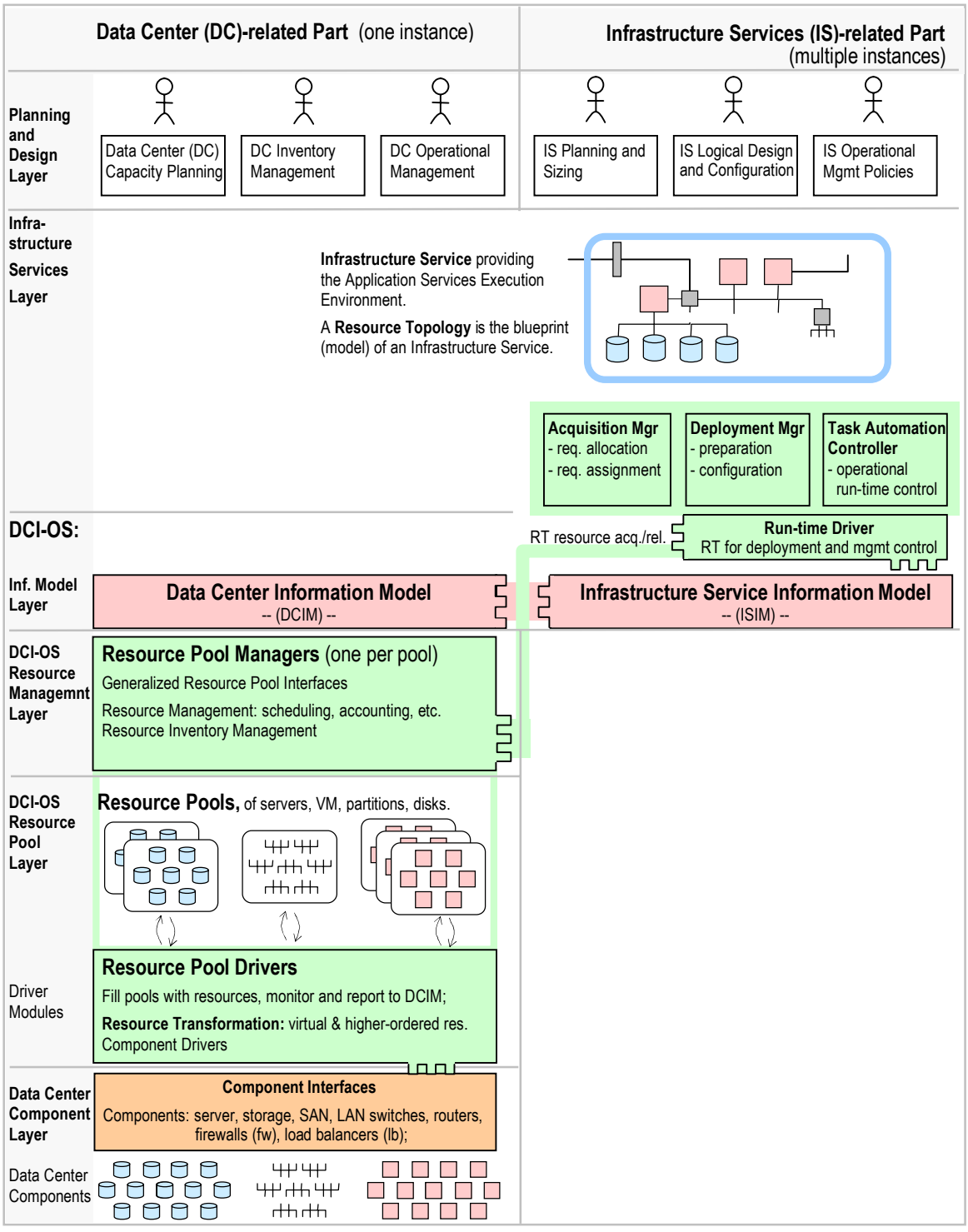


Figure 7: Architecture of a Data Center Infrastructure Operating System (DCI-OS).

The three layers of the architecture are:

- *Planning and Design Layer* – teams of people are involved in planning and designing at both sides, data centers as well as infrastructure services in data centers. People gather, share, exchange and transform information at this layer using a variety of practices and planning and design tools.
- *Infrastructure Services Layer* – comprises an environment of configured and operational resources forming the execution environment for an application service. An Infrastructure Service is the basic abstraction produced by the DCI-OS and can be seen as the analogy to the execution environment an operating system provides to an operating system process.

  A Resource Topology is a blueprint (a model) of the set of resources of an IS along with their configurations and policies which allow automated processes for deployment and management later.

  A Run-time Driver as part of the IS provides the interface to the central DCI-OS modules. A difference to operating systems is that substantially more complex logic is needed for an IS to provide capabilities for acquiring and releasing resources, deploying them and providing task coordination capabilities. This functionality is shown in form of the Acquisition Manager, Deployment Manager and the Task Automation Controller. Furthermore, an information model complements the IS, which is shown as the Infrastructure Service Information Model (ISIM) in the architecture. This information model provides the connection between the IS and the DCI-OS in the data center in which it is deployed.
- *Information Model Layer* – maintains all relevant information of the Data Center Information Model (DCIM) and the models of Infrastructure Services (ISIM), which are known in the DCI-OS. Models include information about plans and designs (the future), as well as information about inventory and present operational conditions. They may also contain information about past conditions (logs, traces, etc.) for audit, accounting or other purposes.
- *DCI-OS Resource Management Layer* – presents an interface to a resource pool, which is a collection of resources of the same type. The Run-time systems of IS can acquire and release resources through this interface. Resource pools maintain information about current and future allocations of their resources.

  The DCI-OS Resource Management Layer consists of a Resource Manager managing Resource Pools, which are another component of this layer. Resource Pool Drivers supply resources into pools as needed. They also perform the complex operation of Resource Transformation when resources needed by pools are of a complex nature and need to be constructed by performing configuration and other operations on more elementary resources. Examples of such complex resource types are virtual machines that are constructed based on physical servers, disks, images on disks and networks in combination with virtual machine software and proper configuration and operational management. Resource construction and resource transformation to create the desired resource abstractions for resource pools is task of the Resource Pool Driver modules.

In operating systems, a similar idea is used to manage machine resources in pools such as disk blocks, memory segments or processor time. Resource transformation processes also exit such as creating a hierarchical file system abstraction based on linear block storage devices.

- *Data Center Component Layer* – is comprised of basic data center components ("raw" resources) that can be transformed into resources for pools by applying proper configurations. The Data Center Component Layer corresponds to the device driver layer in an operating system which task it is to provide generalized interfaces to the variety of component interfaces in computing machines.

The following sections discuss the architecture, its components in more detail. They also discuss where principles were adopted from architectures from operating systems and where differences exit.

5.1 The Planning and Design Layer

The planning and design layer shows people in the data center-related part and the part of infrastructure services. For the data center part, these are teams in a data center IT organization, who are involved in data center planning, design and operational management. These teams of people are responsible for tasks such as capacity planning, inventory and customer management, operational management, facilities, supply chains, accounting and all other aspects of data center management. Here, the focus is on tasks related to Data Center Capacity Planning, Inventory Management and Operational Management.

People at the side of the Infrastructure Service are involved in planning and designing infrastructure services and deploying them into data centers. People include (teams of) solution architects at planning, sizing and design stages as well as IT specialists for subsequent configuration, deployment and operational management of the infrastructure service. Focus here is on Planning and Sizing (sizing is the discipline to determine the resource needs to support an anticipated workload), Logical Design and Configuration, which includes making the right choices based on which resources an Infrastructure Service will be built on and developing the configurations for those resources, and Operational Management Policies, which determines the policies (guidelines) based on which the Infrastructure Service will operate. Examples are expected service levels, maintenance policies, monitoring policies, etc, which need to be determined.

Groups of people on both sides are concerned with complex information. At planning and design stages, it is *only information* that exists and that needs to be gathered, shared and transformed. In today's practice, only rudimentary support is provided for managing this information. In many cases, information is gathered and shared informally as documents and spreadsheets and is communicated among teams via email.

Richer information management systems need to be developed for a DCI-OS enhancing the information from informal to more formal. We use *Models* to refer to formalized information. The lack of formalized information is partially due to the lack of tools supporting architects and specialists on both sides. Lack of use of formalized information prevents that tools are being built. However, over time and with advantages coming from more formalized approaches, the situation will improve. Today, the lack of tools and the lack of formalized (modeled) information remains a major obstacle for improving the

disciplines in the Data Center Planning and Design Layer. Lack of formalized information is also a major inhibitor of subsequent automation leading to the effect that many tasks in data center and application services' management need to be performed manually today.

As part of research, design tools for complex resource topologies and methods for policy-based configuration have been developed aiming at supporting the creation of valid formalized information about data centers and infrastructure services.

5.2 The Infrastructure Services Layer

An infrastructure service is a fully operational execution environment in which an application service can operate. As application service is understood a fully operational application system proving a service of interest, such as a Customer Relationship Management (CRM) application. The application service is comprised of a number of software and data components (application server, data base, etc.) which require a number of properly configured resources underneath, such as servers with the right software installed, load balancers with the right configuration, storage arrays with the proper disks provisioned and images installed, networks that are properly configured allowing all these components interact seamlessly. This is what an infrastructure service provides as the execution environment for an application service. It includes all needed resources (servers, storage, and networks) with proper configurations as well as application software and data made available and operational on those resources.

An *Infrastructure Service is the basic abstraction produced by a DCI-OS.*

The blueprint of an Infrastructure Service is defined by a model which is called a *Resource Topology*, which describes all needed resources along with all necessary configurations in order to make resources function together. Since a Resource Topology is a model, it can be instantiated multiple times leading to multiple infrastructure services that can coexist in one data center or can be deployed in multiple data centers.

Goal of the Resource Topology is to enable and drive *automation* over a large extend of the lifecycle of an Infrastructure Service, from provisioning the resources needed to driving their configuration and deployment through to subsequent operational management.

For this purpose, a number of DCI-OS components need to be provided that allow these functions as part of the IS. These components are:

- *Acquisition Manager* – manages the allocations and assignments of resources from resource pools managed in the data center to an infrastructure service.
- *Deployment Manager* – manages activities related to the preparation and execution of deployment tasks. Deployment is an activity that applies configurations onto resources in order to make them function in context of the IS.
- *Task Automation Controller* – coordinates the workflows of the IS-part of the DCI-OS such as deployment or activation of an infrastructure service. Workflows can be initiated by an operator through a console or can be triggered programmatically through an external interface. The Task Automation Controller operates based on the concept of a closed loop controller that

correlates observed state with desired state. Operators define the desired state in the information model as part of the management policy for the Infrastructure Service.

- *Run-time Driver* as part of the infrastructure service – provide the connection to the managed infrastructure service when it is in existence. The run-time system has two main tasks. First, to determine and report the current state; and second, to execute control operations received from the operational management controller. Examples are basic lifecycle operations such as start, stop, suspend or resume the infrastructure service.

For comparison, one can see the resources provisioned into a running operating system process as an analogy of an Infrastructure Service in operating systems. There is no analogy of the concept of an explicit blueprint or a model of a Resource Topology in the operating system context.

5.3 The Information Model Layer

Systems, such as a DCI-OS, require formalized information in order to operate. A framework is needed in which all relevant information can be represented and managed and also allows driving subsequent automation processes. We use *Models* as such a framework. The Information Model Layer consists of models from the data center and the IS side and comprises a major building block of a DCI-OS.

Operating systems require information as well. But they maintain a light-weight model in form of simple in-kernel data structures such as a list of process control blocks, structures of memory mappings, i-nodes or device descriptors. IT management systems typically maintain an information model in a database such as a Configuration Management Data Base (CMDB) or, in a lighter-weight form, as data structures defined by a Management Information Base (MIB). A number of information modeling frameworks exist, most prominent in system management is the Common Information Model (CIM) defined and standardized by the Distributed Management Task Force (DMTF), which has been a standard in the IT industry for many years.

The *Information Model* of a DCI-OS is shown in Figure 7 as a layer with two connected components: the Data Center Information Model (DCIM) and the Infrastructure Service Information Model (ISIM). Connecting models means that the design of an infrastructure service must take into account which resources are available in a data center at the projected time of its deployment and operation. Resource Topology designs should allow for flexibility in their resource needs (types, quantities) in order to produce working IS under varying resource supply conditions in data centers. Other factors must be taken into account as well. The data center information model exposes current and future *capabilities* of a data center. The infrastructure service information model presents *requirements* of its needs. Exchanging information about both, data center capabilities and infrastructure service requirements, forms the connection between both models. Requirements and capabilities are then interpreted at the other side.

Information models from multiple infrastructure services can connect to the information model of a data center at the same time. This connection occurs early already during planning and design stages and continues during operation. The connection is dissolved

when no information about an infrastructure service needs to be maintained in a data center any more, which can be long after the service has been retired.

In contrast to operating systems as well as IT management systems, the information model for a DCI-OS can not only incorporate information about conditions and elements that *currently* exist in the managed environment, or which have existed in the past, it must also incorporate information about designs and plans, which reflect state and conditions in the *future*. One reason for this need is to support *automation* of the creation of infrastructure services. At the time of creation, the infrastructure service does not yet exist, only information about its planned and designed states exists. A second reason is the *coordination* that is required with the data center when multiple infrastructure services are to be deployed. It must be coordinated ahead of time when which service is scheduled and which resources they are allocated to use. A third reason is the automated *resource pool management*, which primarily relies on forecasted or known information about capacity and capability planned and/or needed in the future for making the right allocation decisions.

Controlling the creation process through models addresses two important problems. First, it allows the system to create the desired environment based on a reusable blueprint, which can be validated to be correct. Second, by doing this, the system not only can observe what is being and what has been created, it also knows what is expected to be created based on the topology design it deploys. Consequently, the deployment process cannot only be automated, it can also be observed and correctness during execution be verified. Failure can be detected and corrected, in some cases automatically as well.

Not knowing the concept of a design (of what should be created) is a major shortcoming of existing IT management systems. They typically lack upfront modeled information about designs and rather discover the managed environment at run-time in order to establish their internal information models. Discovery delivers information about the current conditions, which elements are there and which state they are in. Discovery cannot determine the elements that were expected to be there and the expected states they were supposed to be in. Hence, neither creation can be automated, nor can be validated by the system whether that what is there are indeed the desired elements. Only operators have the contextual knowledge about plans and designs and hence can relate this information to the current state and modify the environment accordingly.

This contextual information operators have today exclusively, particularly about blueprints, designs and future desired states, must be formalized and made available to a DCI-OS enabling it to automate the creation of designs, but also to detect errors and intervene with corrective action is needed as part of automated operational management. Evolving to more formalized models about designs in IT management not only enables automation of the creation process, it also allows automatable verification whether what has been created is in compliance with the design.

Consequently, extending the information model from reflecting only current states to also incorporate future and expected (desired) states in combination with representing designs and plans is a fundamental change that has to occur in IT management systems. A management system must be able to make a determination whether an *observed state* is also the *desired state*. If this information is present, operational management automation can rely on feedback loops which is constantly comparing observed states against what

had been planned, designed and is expected. Automated corrective action can be deduced from differences between observed and desired states.

Operating systems on the other side maintain a number of internal data structures representing the current state of the machine environment. They also incorporate built-in policies for managing resources, such as processor scheduling or memory scheduling. None of the data structures is comparable to the complex information models that are maintained by IT management systems. Reason is that the environment of a single machine, although it can be complex, is largely standardized, the variety of components is known and typically not changing over the lifetime of the machine. The situation in a data center is very different. Components are not known which may become available at some point and the inventory is constantly changing requiring constant adjustments to the data models in the information base.

Consequently, for a DCI-OS, the information model must incorporate information about designs and plans as well as desired and observed states for automating operational management later. Supplementing information models with information about plans and designs as well as desired state is another major enhancement of the DCI-OS information model over other infrastructure automation solutions.

5.4 Resource Management Layer

As the DCI-OS assumes that people provide information input in form of models in the planning and design layer, it also assumes that its technical modules make use of these models and supplementing them with current information obtained from the various systems. The layer below the Information Model Layer consists of DCI-OS modules carrying out tasks needed in the data center part of the DCI-OS to support the various IS deployed in that data center. Modules operate based on the information supplied from the information models.

Following modules constitute the DCI-OS Resource Management Layer:

- *Resource Pool Managers* – manages pools of resources and allocations and assignments of resources from pools to infrastructure services.
- *Resource Pools* – collections of simple or complex (transformed) resources of the same type for purpose of allocation into IS.
- *Resource Pool Drivers* – produce the resources for resource pools. Resources can be basic resources (as they are found as components in the data center) or can be transformed resources created on basic resources.

5.4.1 Resource Pool Managers

Resource Pool Managers provide the capabilities to allocate and assign resources from resource pools to Infrastructure Services. Requests for resources are accepted from Resource Pool Managers via Acquisition Managers as part of Infrastructure Services.

In contrast to an operating system, where requests for resources are typically of a simple nature, e.g. the request for more memory or the request for a new file, usually fulfilled instantly if the resources are available, resource requests in a DCI-OS are more complex.

First, they refer to more complex resource types, which are results of transformation processes in the Resource Pool Drivers. Second, requests can refer to future times and

scheduled accordingly when it is known when resources will be needed. Third, resource requests can be of an abstract nature, such as "an Intel-based machine", allowing the Resource Pool Managers more flexibility in fulfilling the request[2]. And fourth, a distinction is made between an allocation of requested resource quantity and assignments of actual resource instances from the pool to accommodate for change in resource inventory, which is typically not the case in an operating system, but which is typical in a data center.

Resource Allocation means a commitment of a defined set of resource quantity to an infrastructure service for an allocation period. This quantity is subtracted from the resource pool capacity for that period. Allocations are used to coordinate *future resource use* where actual resource instances (inventory) may not be known and may change. Allocation operates based on generalized resource properties such as general type, capacity or quantity and an allocation period. At the time of allocation it is not known which resource instances will be made available at the time of fulfilling the allocation.

Resource Assignment then occurs when the infrastructure service actually needs resources. Individual resource instances are chosen from pools. While resources must exist in pools for assignment, for allocation they need not. An allocation must precede an assignment. An allocation also spans the time of assigned resources, when resources are actually used by an infrastructure service. Reason is that capacity is still bound to the infrastructure service during use. Allocation can only end when all assigned resources have been released. Allocations as well as assignments may be dynamic and change over time. Capacity of assigned instances cannot exceed capacity of allocated instances.

The information about allocations in the future also guides planning and capacity management in a data center since it reflects future demand for resources.

The decoupling and *late binding* between resource allocation and resource assignment is another enhancement over existing resource management systems.

5.4.2 Resource Pools

A resource pool is a collection of resources of the same type. Basic resources can be directly maintained in pools such as pools of physical servers. Resources in pools can also result from transformations performed by DCI-OS resource pool drivers. New resources are created that are based on underlying basic resources. Like in operating systems, two main categories of transformations exist: resource generation and resource virtualization. Resource generation produces resources with new qualities while resource virtualization multiplies one kind of resource by either expanding its capacity of multiplying the number of instances.

Examples of generated resources in an operating system are files and file systems generated in block devices. Examples of generated resources in a DCI-OS are disks

[2] A process called Grounding is applied to map requests for higher-ordered resources into resource types that are available in pools.

created in a storage array or a subnet generated in a network infrastructure. For this purpose, the storage array or the network switches and routers must by properly configured in order to generate the desired disks or subnets. Resource pool drivers as part of the DCI-OS perform this task of properly configuring data center components and by doing this producing the transformed resources that are then managed in pools as well.

Examples of virtualized resources in a data center are virtual machines, which are generated by deploying and configuring virtual machine software on physical servers.

Higher-ordered resources may be generated as well which can be database or application server instances (in their standard configuration) which are also created by DCI-OS resource pool drivers and managed in pools like other resources.

5.4.3 Resource Pool Drivers

Resource pool driver modules produce the resources and add or remove them to pools. Pools of basic or transformed resources are fed and maintained by Resource Pool Drivers, which have the ability to produce more resources of certain type when requested by Resource Pool Managers, e.g. when pools run low in inventory.

The concept of a resource pool is a single abstraction used by the resource pool manager module of a DCI-OS. The resource pool manager module manages capacities of pools and allocations and assignments of resources to infrastructure services from pools.

5.5 Data Center Component Layer

The data center component layer is comprised of the basic data center components as they are found in a data center. These components can directly be used as resources managed in pools (performing a "null" transformation) or they can be used as basis for complex resource transformation by the Resource Pool Drivers.

Particularly for transformation, but not only for this purpose, resources need to be instructed, controlled, managed and configured by the Resource Pool Drivers requiring an interface and a connection to exist.

For most basic resources, these interfaces use standard protocols such as SNMP and WBEM, but usually implement proprietary functionality which needs to be addressed by the Resource Pool Driver responsible for a certain basic resource type.

The analogy to operating systems are device drivers which provide generalized interfaces in established abstractions to device drivers on one side (e.g. the file interface in Unix with operations of open, close, read, write, ioctrl, etc.) and specific interfaces to the device on the other side.

A difference to operating system drivers is that resource transformations are not performed in the device driver layer (e.g., block device to file system). Resource transformations in operating systems in general are of a simpler nature than it is the case for a DCI-OS.

5.6 Summary and Discussion

This section relates concepts used and developed for the architecture of a DCI-OS with concepts known from operating systems and IT management systems and the benefits that result from the synergies leading to its main innovations.

5.6.1 Benefits Resulting from the DCI-OS Architecture

The architecture in Figure 7 significantly draws on concepts known from operating systems. It supports separation of concerns by applying the concept of layers and principles of information hiding and abstraction enabled by layers. This allows a number of benefits that originate from the architecture.

One major benefit is that infrastructure services can be developed and maintained independently as modules from a particular data center and from each other. And vice versa, data center evolution can also better be decoupled from the legacy of applications running in them.

Reusability of infrastructure services, replication and thus packaging as separate modules that are pre-tested are further benefits significantly improving today's practice of tightly coupled application deployments in data centers today.

Modularization on the data center side also allows the data center infrastructure to evolve more independently from the modules of infrastructure services hosting the various application services. The vertical and horizontal boundaries help identifying explicit interfaces and placement of functionality into the proper compartments of the architecture.

The separation of concerns achieved by this architectural pattern is a major enhancement over prior, more monolithic data center automation solutions such as HP's Utility Data Center [UDC].

A significant benefit of the architecture is that it enables automation in driving the processes of transforming resources into types requested and required by application services, managing those resources automatically in resource pools and allowing consuming infrastructure services to allocate, use and release resources as needed by dynamically assigning and binding them to underlying resource instances rather than using static assignments used in data centers today.

5.6.2 Concepts Adopted from Operating Systems

The following list summarizes concepts from operating systems that have been adopted for the architecture of a DCI-OS:

1. The concept of structural layers and modules within layers was adopted as a structural principle to achieve modularization, isolation, separation of concerns as well as the ability to formulate defined interfaces between the technical components of the system.
2. The concept of a common "kernel" shared between all application processes was adopted and reflected in the Data Center-related Part and the part of Infrastructure Services.
3. The Planning and Design layer was added, which is not typically considered in operating systems. This layer also links the DCI-OS system early with the people who are in charge planning and design processes in IT.
4. The concept of a resource was adopted as well as concepts of resource abstraction from
    a. Basic resources into
    b. Transformed resources constructed based on underlying more basic resources providing higher resource abstractions.

5. The concept of a resource was extended by the introduction of a definition in form of a blueprint for a Resource Topology, which is represented as a formal model.
6. The concept of a resource was extended by the introduction of planning and design stages occurring before resources can be requested.
7. The basic idea from operating systems to manage resources as pools was adopted as well as the approach introduce an abstraction layer in form of drivers isolating generic functionality resources provide from underlying vendor-specific interfaces and protocols which need to be accommodated when interacting with those resources in the environment.
8. The concept of requesting resources was refined by introducing a distinction between resource allocation (which refers to a commitment to provide a quantity of a certain resource type at the requested time frames) and assignment occurring shortly before use (late binding of actual resource instances from pools to fulfill an allocation) allowing the system the flexibility to incorporate future resource needs at early stages and making them known in the system as well as dealing with the latent issue of changing inventory in data centers, which does not occur in an operating system.
9. The concept of a run-time system from operating systems providing means for the application service to interact programmatically with the underlying operating system were accommodated and adopted such that requests and releases of complex resources can me formulated and transmitted to the DCI-OS, as well as management functionality can be accessed on own resources.
10. The concept of a run-time system needed to be significantly extended in order to accommodate capabilities to deploy configurations onto resources as well as control complex automated task flows. New components of an Acquisition Manager, a Deployment Manager and a Task Automation Manager were introduced.
11. The concept of "kernel"-data structures in operating systems was significantly extended into a two-fold information model comprised of the Data Center Information Model and the Infrastructure Services Information Models capturing current and future, desired states along with management policies for the data center part and the part of infrastructure services.

5.6.3 Supporting Requirements

The previous chapter provided a list of requirements for a DCI-OS. The following discussion relates the architecture back to these requirements. Discussion was separated into the discussion of requirements supporting aspects of data center management and requirements supporting individual Infrastructure Services and their binding into the environment of a data center.

The first category of requirements related to overall requirements of data center management and their reflection in the architecture of a DCI-OS in the Data Center-related part (or column) in Figure 7.

- **R.110 – R.120:** resource capacity planning and inventory planning is directly supported in the data center planning layer of the architecture of the DCI-OS.

Resource capacity and inventory planning are essential tasks in data center management. They can be supported by tools and methods such as discussed in Chapter 7 (The Planning and Design Layer). A particular consideration is given to the fact that planning refers to a state of future existence and needs to be taken into account over current practice in IT management systems to only know about current resource inventory and use. One consequence of this aspect is that resource capacity must be separated from specific resource types which may not be known. The methods presented in Chapter 7 pay specific attention to this fundamental aspect.

- **R.120 – R.140:** a DCI-OS must create resources as needed by Infrastructure Services. Requirements in this category address fundamental organizational properties of resource abstractions, their categorization and modeling, which are fundamental building blocks of the Data Center and Infrastructure Services Information Models forming the Infrastructure Model Layer in the architecture of the DCI-OS. In the particular realization, the Common Information Model (CIM) [CIM], an established modeling framework in IT Management by the Distributed Management Task Force (DMTF) [DMTF] was used to represent models for both sides, the Data Center Information Model and the Infrastructure Services Information Model. Chapter 8 (The Infrastructure Services Layer) provides a more detailed view on how the abstractions were represented in the CIM modeling framework.
- **R.140 – R.170:** address fundamental aspects of resource organization in a data center in an abstraction of pools from which resources can be acquired and released back. Resource Pools also form the fundamental abstraction of resource management in the architecture of the DCI-OS and is supported by the DCI-OS Resource Management Layer and the DCI-OS Resource Pool Layer.
- **R.170 – R.190:** describe requirements that management of resources in pools is automated like in an operating system. These requirements are addressed in the architecture by the Resource Pool Managers which are software components which automatically manage resource pools, accept or deny requests for resources from pools for Infrastructure Services and in turn request or release resources back into the data center environment.
- **R.190 – R.200:** address the requirement that for automated management resource components in a data center must be programmable, which is addressed in the architecture by the Resource Pool Driver modules and the Component Interfaces providing access to the actual physical data center resource components.
- **R.200 – R.220:** these requirements reflect that resources must be discovered automatically in the data center environment as well as their use must be monitored and accounted. Both requirements are supported by Resource Pool Managers in the architecture of the DCI-OS which perform both functions.

In summary, the following requirements for the side of data center management are supported by the architecture of a DCI-OS:

- Support for capacity management and data center planning.
- Ability to deal not only with current inventory, but also with projected future inventory.
- Manage capacity rather than inventory, particularly in the future.
- Single abstraction for data center resources: *Resource Pools*.
- Automated resource pool management.
- Support for transformed resources, specifically virtual resources in pools.
- Support for sharing resources.
- Exclusive access to data center components through programmatic management interfaces eliminating direct human access to data center components.
- Automated discovery and consolidation of current with projected inventory in the data center information model.

The second category of requirements related to Infrastructure Services and their reflection in the architecture of a DCI-OS in the Infrastructure Services-related part (or column) in Figure 7.

- **R.300 – R.400:** address aspects of descriptions (designs) of the resource environment needed by an Infrastructure Service. In the architecture, this is represented in the building blocks of Application Planning and Sizing and the Logical Design and Configuration in the Planning and Design Layer. The basic abstraction of a Resource Topology reflects this set of requirements. Expressions of policy-based management have the purpose to express operational management policies (R.360 – R.370). The also have the purpose of expressing rules of refinement of higher-ordered expressions for resources as part of a Resource Topology allowing their automated refinement into particular resource types, a process which was introduced as Grounding. Chapter 7 has a detailed discussion of the realization of these advanced automated techniques.
- **R.400 – R.500:** reflect requirements for automated resource acquisition and release by Infrastructure Services, which is supported in the architecture of the DCI-OS by the Acquisition Manager in the Infrastructure Services Layer, which is part of the run-time environment of an Infrastructure Service connecting it to the Resource Pool Managers in the particular data center where the Infrastructure Service is deployed.
- **R.500 – R.600:** describe requirements of automated deployment of Infrastructure Services into data center environments, which is the task of the Deployment Manager of the Infrastructure Services Layer. Its task is to create (~deploy) the settings needed by Infrastructure Service in order to create an instance. For this, the Deployment Manager, as part of the run-time environment of the DCI-OS for an Infrastructure Service, interacts with the acquired resources deploying the configurations from the Resource Topology of the Infrastructure Service. Having explicit models of those configurations allows for automation of deployment

processes, which has been one of the key motivators behind a DCI-OS: data center automation.

- **R.600 – R.700:** addresses requirements referring to the ability of controlling and coordinating the automated deployment processes on the resource set assigned to an Infrastructure Service. In the architecture of the DCI-OS, this task is provided by the Task Automation Controller, which is part of the run-time environment of the Infrastructure Service Layer.
- **R.700 – R.900:** describe requirements for automated operational control, such as lifecycle control over Infrastructure Service instances. These tasks are also provided by the Task Automation Controller, which implements a declarative approach rather than a process-oriented approach by providing expressions of a "Desired State" to which the controller constantly observed the "Current State" of the associated Infrastructure Services components keeping them in alignment of what has been declared as Desired State. Chapter 9 (The DCI-OS Layer) provides a detailed description of the realization of such a Task Automation Controller.

In summary, the following requirements are supported by the architecture of a DCI-OS for the side of Infrastructure Services:

- Model-driven design and management based on a common abstraction: Topology.
- Modularity and reuse of topology designs.
- Openness and modularity of model layer.
- Multi-instantiation and multiplication of topology designs in a data center.
- Portability of a topology design into other data centers.
- Support for transformed resources, specifically virtual resources.
- Isolation between resource specifications and bindings to actual resource instances; late binding between resource allocations and resource assignments.
- Apply formal methods to ground and size a topology for requirements.
- Automated resource acquisition.
- Automated deployment (concept of an automation controller).
- Automated, policy-driven operational management including automated correction of certain error conditions.

Chapter 6

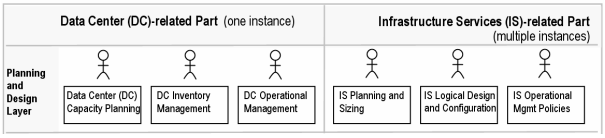
Research, Realizations and Case Studies

The purpose of this chapter is to provide an overview of the research, experiments, realizations and case studies that have been conducted between 2002 and 2007 in context of the architecture of the DCI-OS. The results of this research have been documented in a number of publications over the years. Some are documented here.

Figure 8 relates the layers of the architecture to the discussions in the following chapters.

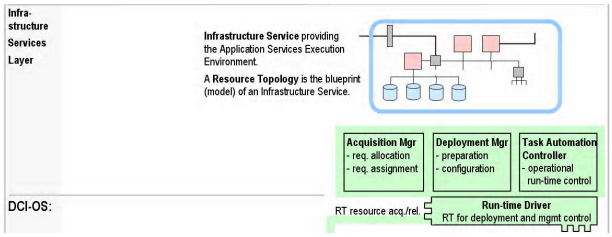
Chapter 7: The Planning and Design Layer

- Performance Engineering for Data Centers (7.3)
- Systematic Approach to IT Configuration (7.4)
- Resource Topology Design (7.5)
- Policy-based Configuration (7.6)



Chapter 8: The Infrastructure Services Layer

- Task Automation Controller (8.1)
- Deployment Manager (8.2)
- Resource Acquisition Manager (8.3)



Chapter 9: The DCI-OS Layer

- DCI-OS Information Model (9.3)
- Resource Management Sub-Layer (9.4) with:
  - Resource Pool Manager (9.4.1)
  - Resource Request Workflow (9.4.2)
  - Resource Allocation (9.4.3)
  - Resource Assignment (9.4.4)
- Data Center Component Sub-Layer (9.5)

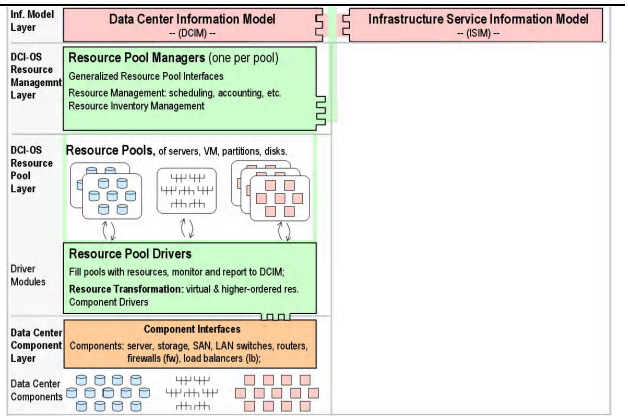


Figure 8: Discussion of research in context of the layers of the DCI-OS architecture.

The architecture of the DCI-OS also guides the discussion of the following chapters addressing research, realizations and case studies which have been conducted. Chapter 7 will address research related to *The Planning and Design Layer*, which is the top-layer of the architecture. Chapter 8 will address the next layer, the *Infrastructure Services Layer*, and chapter 9 presents the *DCI-OS Layer*, according to the architecture.

Research and prototypes have been documented in a number of publications over the years providing proof points and validation. This chapter presents a broad overview of research topics which have been addressed. A selection of this research which involved the author is then discussed in more depth in subsequent chapters.

It should be noted that the research and prototyping work occurred *before* the architecture described in this thesis was developed. The architecture presented in chapter 5 is the result combining a number of research efforts towards data center automation into an architectural framework guided by concepts from operating systems and IT management.

6.1 Overview of the Research

The research was conducted under the name of a research program called *Quartermaster* in HP Labs. An overview of the architecture, purpose and goals can be found in the IM 2005 paper *"Quartermaster – A Resource Utility System"* [Sing05], [Sing04].

The conceptual architecture of this system was documented in *"Conceptual Architecture of Quartermaster"* [Gra03b] and is presented in section 9.3 describing novel concepts and abstractions developed in the Information Model for the DCI-OS. This work formed the core part of Quartermaster since it developed the fundamental abstractions of the information model as entities and relationships the DCI-OS has to deal with in order to perform its functions.

6.2 The Planning and Design Layer

The Planning and Design Layer addresses issues of pre-operational stages from the data center perspective and from the perspective of infrastructure services.

In a data center, planning, deploying, managing and improving data center resources is a continuous process. Capacity planning is the process of estimating future needs for resource capacity and making decisions about acquisition plans and their implementation, including facilities, power and supplies for the data center.

Research for the Planning and Design Layer is presented in detail in chapter 7.

6.2.1 Performance Engineering for Data Centers

There has been a long tradition of research in HP Labs around performance engineering and workload characterization and prediction. Research was applied to a number of systems and in context of Quartermaster further developed into techniques for aggregated demand characterization and prediction for shared IT infrastructure.

Papers resulting from this research have been "Capacity Management and Demand Prediction for Next Generation Data Centers" [Gma07], "A Capacity Management Service for Resource Pools" [Rol04], "Capacity Management for Adaptive Enterprise Resource Pools" [Rol05], "A Composite Framework for Application Performability and

QoS in Shared Resource Pools" [Cher06] or "A regression-based analytic model for dynamic resource provisioning of multi-tier applications" [Zhang07].

Basis of discussion in section 7.3 forms a paper "APE: An Automated Performance Engineering Process for Software as a Service Environments" [Rol08].

Part of the research has been in collaboration with the University of Munich which resulted in a PhD dissertation *"Managing Shared Resource Pools or Enterprise Applications"* [Gma09]. Recent and ongoing work addresses problems of *"Integrated Capacity and Workload Management for the Next Generation Data Center"* [Zhu08] and *"An Integrated Approach to Resource Pool Management: Policies, Efficiency and Quality Metrics"* [Gma08].

Early work on data center modeling which also incorporated energy awareness was jointly addressed with the smart data center team and has been documented in *"The Smart Data Center"* [Pat03] and *"Energy Aware Grid: Global Workload Placement Based on Energy Efficiency"* [Pat03a].

6.2.2 Systematic Approach to IT Configuration

Goal of this work was to develop automated approaches to derive IT configurations from higher-level specifications such as business processes supplemented with non-functional requirements such as expected numbers of users determining the "size" of an application configuration as input for configuration generation tools as part of the design process of applications before deployment. There are a number of well understood processes such as the application sizing and reusable best practices templates. However, these are isolated and their tool chains not integrated.

In a joint collaboration with SAP Research, an integrated approach has been developed using a model-driven approach. Basis of section 7.4 is a paper describing the approach for *"Deriving IT Configurations from Business Processes"* [Gra08] and *"Business-driven IT for SAP – The Model Information Flow"* [Bel07]

Prior work is related to policy-based design and configuration generation techniques, which have been investigated as part of the research program.

6.2.3 Resource Topology Design

One of the core abstractions developed during research is the concept of a Resource Topology, as a model of a configured environment of IT resources. A Resource Topology included a set of instances of resource types with their relationships, such as network connections. A Resource Topology is an analogous model to a model of connected circuits used in hardware design, but applied to resources used in data centers.

With the development of the concept of a Resource Topology, models of resource environments can be designed by application architects reflecting the resource needs application services instances have in a separate design step, independently of a data center environment where the Resource Topology would be deployed.

The separation of design and implementation of resource configuration, which has not occurred in the IT practice today, allows the creation of explicit model representations of Resource Topologies as part of the Information Model of the DCI-OS. It allows reuse in terms of repeatable deployment of resource configurations, significantly accelerating the today manual deployment process.

Introducing Resource Topologies as designs (models) of configured resource configurations also allows (and requires) the creation of graphical design tools, which was one part of research. Since Resource Topologies (as models) allow the explicit representation of a configured resource environment, another research domain addressed the automated generation of Resource Topology models, which was called Policy-based Configuration.

The main work for the graphical design tool was published under *"Policy-based Resource Topology Designer for Enterprise Grids"* [Gra05], which also forms the basis of discussion in section 7.5.

Another design tool was developed as a prototype for infrastructure (resource) environments for SAP standard deployments, which has been described in *"ModelWeaver – A Service Design Environment for SAP"* [Gra08a].

6.2.4 Policy-based Configuration

Policy-based configuration referred to a line of research to programmatically generate IT configurations, enabled by the presence of explicit models of Resource Topology designs. A number of approaches were explored incorporating constraint satisfaction solvers. The approach finally chosen was to capture a number of feasible configuration patterns for known standard configurations, e.g. for standard SAP deployments, in form of templates. A number of templates could exist for a configuration reflecting various sizes or target architectures. For choosing a specific template, policy information is used provided as input by a solution architect. Policy information then also guided the refinement process for the chosen template to generate the final Resource Topology.

The selection and refinement steps were approached with a constraint satisfaction solver operating on input which was transformed from models.

A number of publications document this research area. Basis of discussion in section 7.6 is the paper *"Policy-based Resource Topology Design for Enterprise Grids"* [Gra05].

Further work included "Cauldron: A Policy-based Design Tool" [Ram06], "Using Object Oriented Constraint Satisfaction for Automated Configuration Generation" [Hin04], "Automated Generation of Resource Configurations through Policies" [Sah04] and "Automated Policy-Based Resource Construction in Utility Environments" [Sah04a].

6.3 The Infrastructure Services Layer

The Infrastructure Services Layer provides support for Infrastructure Services. Its capabilities include acquiring and releasing resources from the data center environment through the Acquisition Manager, support for deployment tasks by the Deployment Manager on acquired resources (deployment of specific configurations for the Infrastructure Service) and the overall coordination of service lifecycle operations provided by the Task Automation Controller.

Research for the Infrastructure Services Layer is presented in detail in chapter 8.

6.3.1 Task Automation Controller

Typically, task automation in IT has been addressed by programming or scripting tasks and, for more developed environments, specification and execution of workflows.

HP has a long tradition in building IT automation systems and solutions based on workflows. The Utility Data Center was discussed in section 4.1 as starting point for the requirement analysis. A major problem with workflow-based automation systems in IT is that they usually describe the flow of successful operations, which is called the "happy path". As long as operations execute without failure, the workflow can continue and produce the desired result at the end without human involvement.

When operations in executing workflows fail, however, or do not succeed in desired time frames, or report success without actually succeeding in the managed environment, then a significant problem occurs of how to handle these situations. Capturing failure cases in a workflow specification makes the specification complex and also only can reflect those cases when the failure of an operation is known ahead of time and has been considered in the workflow specification.

Usually, the workflow terminates on occurrence of a failure. While this is sufficient from the perspective of the workflow engine, it does not affect the state changes that have been made to the managed environment as result of prior succeeding operations. For example, disks might have been created in a storage array as result of a succeeding step, but a subsequent step of attaching the LUNs for those disks to server machines fail stalling the workflow execution. In this case, the disk creation operation has permanently altered the state of the disk array and must be rolled back in order to clean up. Achieving roll-back semantics with workflows is difficult since the roll-back logic must be fully-specified in the workflow with explicit statements.

Another problem with workflow-based automation in IT is the asynchronicity of operations due to the fact that most operations on IT resources require long execution times, e.g. copying an image onto a disk requiring minutes to complete. Components also may fail after they have succeeded, e.g. a sudden server crash after the server has been deployed successfully by a workflow. In this case, the deployment operation rightfully reported success such that the workflow can continue, but a failure occurs with the server later on. In workflows, there is no appropriate mechanism to catch and handle asynchronous events from the managed environment after operations have succeeded.

All these problems have caused substantial difficulty in creating robust task automation systems based on workflows. The Utility Data Center was one example allowing automation as long as all operations succeeded, but required manual interference to clean up the environment after failures causing significant difficulty (and time required) by operations staff to determine the state in the managed environment that has been altered and removing this state cleanly. One operator stated that this difficulty often required more time than manually deploying a configuration in the first place avoiding automation workflows altogether.

For these reasons, a research thread was initiated to consider alternative and more robust automation patterns. The research led to a controller pattern, which also formed the basis for the realization of the Task Automation Controller implementation. Research and realization of Task Automation Controllers has been published in a number of papers and led to a number of patents.

In contrast to a workflow, a controller pattern allows the declarative specification of a "Desired State" against which a currently "Observed State" is constantly compared and actions are initiated when elements of the desired and the observed state differ. This way,

the controller continuously aims to keep the managed environment (reflected by the observed state) in alignment with what has been set as desired state. The controller simultaneously observes events reported from the managed environment and can also respond in parallel to observed state changes occurring in multiple managed elements.

Basis of the detailed presentation of the Task Automation Controller in section 8.1 is a paper published at IM'2007 *"Automation Controller for Operational IT Management"* [Gra07]. This paper describes a realization of a controller using a Petri-net description and a Petri-net execution engine which was implemented for this purpose.

Earlier architectural work on automation controllers has been developed and described in an architectural specification *"Specification of the Service Delivery Controller (SDC)"* [Col05] and published in [Tho05].

6.3.2 Deployment Manager

The Deployment Manager allows applying specific configurations from the Resource Topology model of an Infrastructure Service environment onto resources. This capability requires an interface which can interact with the resource(s) and a capability to interpret the model of a Resource Topology.

Basis for the discussion in section 8.2 is a specific deployment system which has been used from HP's product portfolio. This deployment system was called Radia Deployment Manager and later renamed into OpenView Configuration Manager (OCM).

In order to use this system in context of a DCI-OS, an adapter has to be implemented to perform necessary transformations between the DCI-OS Resource Topology models and the data models Radia required.

Since Radia has been an existing management system fulfilling a particular task in context of a DCI-OS, it was also explored under the perspective of integrating existing management systems and tools into the framework of a DCI-OS. Research addressed interface and data integration as well as transformation of data models.

Integration aspects were addressed using web services management middleware (specifically OGSI [OGSI]) and are described in *"Model-driven Software Configuration With the Radia SDC"* [Gra05b]. Necessary extensions and an encapsulation turning the Radia Deployment Manager into a component following the controller pattern have been documented in *"Extending Radia Into A Service Delivery Controller"* [Gra05a].

Earlier work on resource and service deployment included "Massive Deployment of Management Agents in Virtual Data Centers" [Gra01] and "Resource-Sharing and Service Deployment in Virtual Data Centers" [Gra02].

6.3.3 Resource Acquisition Manager

The Resource Acquisition Manager is another part of the Infrastructure Services Layer. Its task is to provide the access to the data center's resources, acquire and release them dynamically as needed. Dynamic resource acquisition in data centers has been difficult in the past in contrast to operating systems, where most resources are requested and allocated dynamically. Reason for this situation is that data center resources required further manual deployment steps in order to be usable in the context of an Infrastructure Service. With the automation of these steps, data center resources can now also be requested and allocated more dynamically.

From the perspective of an Infrastructure Service, two main operations are important, the acquisition of resources when needed (planned or dynamically on demand) and the release of resources.

The Resource Acquisition Manager also follows a controller pattern where a desired level of resource supply is constantly compared with the actual resource usage consumed by the Infrastructure Service.

Basis of the discussion in section 8.3 are papers "Adaptive Control System for Server Groups in Enterprise Data Centers" [Gra03a] and "Adaptive Control for Server Groups in Enterprise Data Centers" [Gra04c] which demonstrated the controller for a specific resource type server. The process of dynamically adding and removing servers based on workload was called server flexing.

The aspect of integrating this controller into the environment of a DCI-OS using standard web services-based management middleware is documented in *"Using HP's Web Services Management Framework for Adaptive Control"* [Gra04].

6.4 The DCI-OS Layer

The DCI-OS layer forms the "core" part for the DCI-OS. According to the architecture of the DCI-OS, it includes the Information Model layer, the Resource Management Layer and the Data Center Component Layer. Its purpose is to produce and manage the resources in the data center and making them available to the Infrastructure Services requesting them.

A number of related research threads have been pursuit in order to explore and define the foundations of this layer.

6.4.1 DCI-OS Information Model

The Information Model is the core part of the DCI-OS layer. It has turned out that substantial complexity arises in the Information Model Layer with the enhanced capabilities this layer provides. For example, the capability to manage not only current resource inventory in the data center, but also information about availability of future resources and making them available for advanced allocation creates the need to manage a time domain with the resources. Furthermore, in order to extend resource management from resources which exist as physical devices (e.g. physical servers) to resources which can be created (e.g. virtual machines, software infrastructure stacks), the concept of resource constructions had to be introduced capturing the mapping relationships of constructed resources onto physical resources. And, as a final example, the capability to automate creation and deployment processes with resources required the introduction of descriptive information for final resource environments, called Resource Topology (model), which comprised the set of resources needed for construction including their connection relationships and the presence of configuration information in a Resource Topology.

These new and complex information types also lead to the need to define complex resource request formats for Resource Topologies.

In order to achieve the desired reusability of Resource Topologies for repeated deployment of standard configurations in the same or in different data centers caused the need to abstract and decouple resource and topology descriptions from actual physical

inventory present in a particular data center in form of (abstract) quantities of resources and connection types. Explicit steps were required to allocate quantities in particular data centers and bind them to specific instances at a late stage before actual deployment.

Fundamental conceptual work has been done by the author of this thesis around those concepts creating the conceptual foundation of the Information Model of the DCI-OS and, through it, the foundations for achieving the technical capabilities that have been desired from the DCI-OS and which were described in detail in chapter 4.

This fundamental conceptual work is independent from the specific choice to select the Common Information Model (CIM) [CIM] as a standardized information modeling framework to represent the desired abstractions in a particular technical framework.

CIM models were defined for the basic abstractions at the meta-class, class and instance levels, which are presented in detail in section 9.3.

The fundamental conceptual framework for the Information Model of the DCI-OS has been defined in a detailed architectural document: *Graupner, S., Singhal, S.: "Architecture Concepts of the Quartermaster Resource Utility System", Conceptual Information Model, architecture document, May 7, 2003* [Gra03b], which also forms the basis for discussion in section 9.3.

A number of modeling and support tools have been built around this methodology including the Resource Topology Design Editor in 2003 [Gra04a], the Quartermaster/CIM Model Browser and Constraint Satisfaction engine as well as the ModelWeaver in 2007 [Gra08a]. Both, the OpenPegasus CIM repository [Pegasus] and the SNIA CIM repository [SNIA-OM] have been used as CIM Object Manager (CIMOM) implementations.

All DCI-OS modules and tools have been built based on the CIM information repositories.

6.4.2 Resource Management Layer

The Resource Management Layer is a sub-layer of the DCI-OS Layer. In itself, it consists of sub-layers of:

- Resource Pool Managers and
- Resource Pools.

Its main function can be described by a Resource Request Workflow, which consists of stages of creating a request for resources by an Infrastructure Service, allocating resources as quantities of abstract resource types and later assigning resources instances as physical resources or constructed resources to these allocations before deployment steps can be applied to the resource environment finalizing it for the requesting Infrastructure Service.

6.4.2.1 Resource Pool Manager

The Resource Pool Manager layer is described in section 9.4.1 and based on work published in a number of papers, including *"Managing Shared Resource Pools for Enterprise Applications"* [Gma09], *"An Integrated Approach to Resource Pool Management: Policies, Efficiency and Quality Metrics"* [Gma08], *"A Capacity Management Service for Resource Pools"* [Rol04], *"Capacity Management for Adaptive*

Enterprise Resource Pools" [Rol05], *"Supporting Application QoS in Shared Resource Pools"* [Rol06] and *"Specifying and Monitoring Guarantees in Commercial Grids through SLA"* [Sah03].

6.4.2.2 Resource Request Workflow

The Resource Request Workflow is described in section 9.4.2 as part of the Resource Management Sub-Layer. Important concepts have been developed, refined and implemented during this work such as the concepts of a complex resource request format, the concept of resource demand and capacity profiles as well as the concept of an allocation calendar, able to not only manage individual allocations of single resource types, but also dependencies between multiple allocations of multiple resource types as part of a resource request for a Resource Topology.

The work section 9.4.2 is based on is the result of a student internship and a following Master's Thesis: *Ralf König "Resource Management for a Resource Utility System"* [Kön04].

6.4.2.3 Resource Allocation

The section on the Resource Request Workflow also mainly addresses the two stages of requesting resources: Resource Allocation and Resource Assignment, which are discussed in sections 9.4.3 and 9.4.4, respectively.

The Master's thesis [Kön04] credited above form the basis for section 9.4.3 as well as other work such as *"Predicting Resource Demand in Dynamic Utility Computing Environments"* [Andr06].

Section 9.4.3 specifically addresses Resource Allocation.

6.4.2.4 Resource Assignment

Resource Assignment is another step in the Resource Request Workflow which identifies actual resource instances in the data center (or fractions of them, or constructions) and binds them to the so-far abstract quantifications of resource types managed in committed resource allocations.

In research, resource assignment was largely considered a selection or placement problem, which was investigated also for optimization, meaning the optimal placement of resource requests onto resources or the optimal selection of resources for resource requests. A relatively large body of optimization work has been conducted such as *"Optimal Resource Assignment in Internet Data Centers"* [Zhu01], *"Algorithms for Self-Organization and Adaptive Service Placement in Dynamic Distributed Systems"* [Andr02], *"Resource Assignment for Large Scale Computing Utilities"* [Zhu03], *"Policy-based Resource Assignment in Utility Computing Environments"* [San04], *"RAMP-A Solver for Automated Resource Assignment in Computing Utilities"* [San05] and *"Adaptive Service Placement Algorithms for Autonomous Service Networks"* [Gra05c].

Section 9.4.4 specifically addresses Resource Assignment.

6.4.3 Data Center Component Layer

The Data Center Component Layer is also a sub-layer of the DCI-OS Layer. In itself, it consists of sub-layers of:

- Resource Pool Drivers and
- Component Interfaces.

6.4.3.1 Resource Pool Drivers

Resource Pool Drivers represent software modules in the DCI-OS which are executing on dedicated management machines and which interact with associated physical components in the managed data center environment via programmable APIs. Resource Pool Drivers apply component-specific configurations onto their associated data center components as well as issue control instructions. In reverse direction, the managed component reports back monitoring data and events indicating state changes in the component.

Resource Pool Driver implementations are highly dependent on the type and model of the associated physical data center components. Issues of compatibility and interoperability, versioning as well as recoverability from failures or unknown states need to be addressed in this layer, which are more engineering issues than research challenges.

[Gra04] describes a Resource Pool Driver which made server resources from the Utility Data Center [UDC] available as Grid resource via a standardized interface. Other implementations of Resource Pool Drivers, which were developed in part by product groups, were not published. There were realized as part of the efforts described below in section 6.5 about Case Studies which have been realized to validate the research.

6.4.3.2 Component Interfaces

Over the years, a number of standards have been proposed and developed for interacting programmatically with managed components. Purpose of a Resource Pool Driver is to interact with the associated physical components in the data center via management the specific management protocols this component supports.

Section 9.5 addresses standard management patterns and protocols, including the discovery of managed components in the data center (section 9.5.1), standardized management protocols such as SNMP [SNMP] and WBEM [WBEM] (section 9.5.2) and newer interface implementations using web services standards such as OGSI [OGSI] management (section 9.5.3).

A broader coverage of web services-based management standards was documented in the book *"Web Services in the Enterprise: Concepts, Standards and Management"* [Gra04b].

Other publications addressed the broader domain of web services-based middleware used for management purposes. Reason for this broader investigation has been that Hewlett-Packard's newer system management product lines should be based on open middleware platforms rather than proprietary middleware implementations.

As part of the research, a number of specific investigations have been conducted related to specific web services management standards and frameworks which had become available at the time. Those were related to standardization efforts in OASIS leading to the standards for *Web Services Distributed Management (WSDM)* [WS-DM], *Web*

Services for Management (WS-Management) [WS-MAN], the *Web Services Resource Framework (WSRF)* [WS-RF] and (earlier) *WBEM: CIM-XML* [xmlCIM].

Implementations were developed using the publicly available Globus Toolkit [GT4]. The work has been documented in *"Web Services-based Management for Adaptive Control"* [Gra04e], *"Management Middleware for Enterprise Grids"* [Gra06] and *"Platform for Delivering IT Management Services"* [Gra06a].

6.5 Case Studies

A number of case studies have been conducted using the techniques developed from the research. Case studies mainly served the purpose to validate research, receive feedback, but also create ideas and new approaches. Case studies were jointly conducted with business units in HP, HP's partners and with customers.

All case studies have been implemented and demonstrated internally to business units and to customers. Two of the case studies continued to productization. Experiences and learned lessons provided a valuable opportunity to guide the research.

Case studies have partially been documented and published. They are briefly listed in the sections below, but not further detailed in this thesis.

6.5.1 Adaptive Infrastructure for SAP

Purpose of this collaboration with SAP was the connection of SAP's Flexible Computing Controller to HP's Blade Automation Infrastructure. The Flexible Computing Controller allowed fine-grained reporting of transaction processing times and workload measurements on performance critical components of a SAP solution, which could be reported through an external interface. At the same time, the controller was able to deploy SAP components onto newly supplied server machines expanding processing capacity of its web server and application tiers. HP's Blade Automation Infrastructure allowed acquiring and releasing server machines on request.

The task of this case study was to create a supervising controller to coordinate both, HP's and SAP's components such that they interplayed properly to achieve dynamic server resource provisioning based on workload measured and reported from the SAP system. In this case, the SAP system also deployed the SAP components onto the newly acquired servers.

This case study has been documented in *"Adaptive Infrastructure Meets Adaptive Applications"* [Edw07], [Bel07a]. The case study has been presented at HP's TechCon Technology Conference 2007 and at SAP's Technology Conference TechEd in 2007.

6.5.2 Operational Management Controller for Oracle Application

This case study was similar to the one before, but with an Oracle database deployment and a cooperating team form Oracle. It preceded the engagement with SAP. Its goal was to demonstrate the integration of management systems and solutions from HP and Oracle to achieve a coordinated behavior of dynamic resource provisioning under fluctuating workloads.

Main challenges were the integration aspects of systems from both sides and the coordination of their interactions. The integration aspects were addressed by using the (at

that time newly released) web services management middleware OGSI [OGSI] and creating OGSI adapters as wrappers around HP's and Oracle's systems. The coordination aspect was addressed by creating those adapters based on the controller pattern, which was a novel approach to an integration problem.

Up until this point, the controller pattern had been applied to the typically numerical control problems such as adjusting server capacity based on workload measurements. It had been a major milestone to extend the controller pattern beyond numerical ranges to discrete state spaces and develop a methodology of how correcting actions can be derived from differences between desired and observed state spaces. The approach used Petri nets and is described in detail [Gra07]. It forms the basis for the Task Automation Controller presented in section 8.1.

The case study for the Operational Management Controller for an Oracle application was documented in "IT Utility Services Using Model-based Automation, Service-Oriented Architecture and Grid" [Cook06] and "Applying Service Delivery Controller in a Blade Automation Case" [Cook06a]. The case study has been presented as a proof-of-concept at HP's TechCon Technology Conference 2006 and at OracleWorld in the same year.

6.5.3 Automated Management of Virtual Desktop Solution

This case study was conducted with a large financial services customer. Goal was to demonstrate an integrated management solution for a large-scale virtual desktop deployment. A virtual desktop deployment removes physical access of employees to desktops or laptops and rather only allows a remote terminal to a desktop session which actually runs on a data center machine. Since desktop sessions usually do not incur high workloads, one server machine in the data center can host a number of virtual desktop sessions, each executing in a separate virtual machine.

Goal of this case study was to demonstrate fully automated management of dynamic provisioning of desktops and freeing data center servers for other deployments when demand for desktops declined, mainly over night in the main working time zones. During those times, financial simulations were run as batch jobs on the same set of physical machines. A number of challenges had to be overcome. One was to avoid long provision times for new desktop sessions when users were login into their virtual terminals. Pools of preloaded sessions had to be created ahead of the typical beginning of working days such that sufficient preloaded virtual desktops were available before people entered their workplaces. With logging in, user-specific profiles had to be deployed in the otherwise identical desktop environments, which required the coordination of a number of systems from outside the solution from user management, to file servers from which user files were accessible. Windows customizations had to be applied such as the user's favorite background screen and other preferences.

The focus of this work mainly was on the integration and automation aspects.

This work has been documented in "Virtual Desktop Initiative: Desktops as Services in a Utility Computing Environment" [Sah05] and "Virtual Desktop System: Consolidating Enterprise User Desktop" [Sah06].

6.5.4 Flexing Interface and Controller for Blade Server Automation

Purpose of this case study with the HP blade server automation business unit was to demonstrate the feasibility of the controller pattern as an approach to solve the problems with workflow-based automation, which was discussed before.

In this case study, a server flexing controller was built which allowed dynamically increasing and decreasing the number of servers which were actively participating in a three-tier standard web application. Adding a new server was triggered when the observed workload passed an upper threshold. It required assigning a server from a pool and deploying the proper configurations onto this server allowing to server to act as part of the web-server or application server tier. When the server was ready, configurations had to be applied to the application environment, such as the load balancer, making the server part of the application environment and serving workload. Reversely, when the observed workload fell below a lower threshold, a server was un-configured from the application environment, wiped clean and returned to the pool of available servers. When the pool of available servers filled above a certain level, servers were powered down in order to save energy.

This earlier case study was described as part of the paper *"Adaptive Control for Server Groups in Enterprise Data Centers"* [Gra04c].

The following chapters will present more detail about the layers of the DCI-OS. Chapter 7 will address research related to *The Planning and Design Layer*. Chapter 8 will address the *Infrastructure Services Layer*, and chapter 9 presents the *DCI-OS Layer*.

Chapter 7

The Planning and Design Layer

The Planning and Design layer addresses issues of pre-deployment stages from the data center perspective and the perspective of infrastructure services.

In a data center, planning, deploying, managing and improving data center resources is a continuous process. Capacity planning is the process of estimating future needs for resource capacity and making decisions about acquisition plans and their implementation, including facilities, power and supplies for the data center. Inventory management is critical to maintain an oversight of the acquired resources in the data center, not only for IT management purposes, but also for financial accounting and reporting. Most inventory management approaches consider current inventory and records about past inventory. Future inventory is rarely considered. Data center capacity planning has been explored as a research area and results are discussed in more detail in this chapter.

Operational management in data centers is a continuous process of managing the operation of a data center. For inventory and operational management, practices have been developed over the years which are well understood and documented in best practices such as ITIL. They have not been explored as part of research in more detail and are thus not discussed in this chapter.

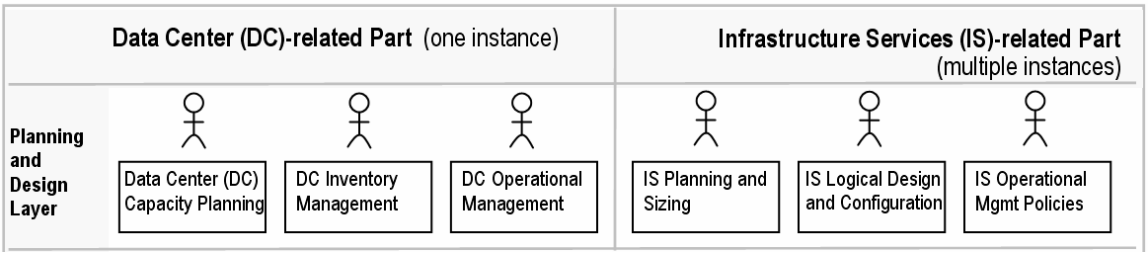


Figure 9: The Planning and Design Layer.

On the side of Infrastructure Services, planning and design tasks are needed as well. These are tasks primarily performed by solution architects who are familiar with the requirements from the application side as well as choices of implementations. *Sizing* is a process performed by solution architects in which workload requirements for an application are translated into tested and documented system configurations. Sizing guidelines are regularly published by vendors of hardware and software systems, often

based on benchmarks. These published configurations describe workloads, such as an expected number of users simultaneously accessing a system. Applications and infrastructure services must be sized to match customer expectations with published configurations in order to make the appropriate choices.

Quantitative analysis and decision making help understand the resource capacities needed for an infrastructure service, but they are not sufficient for determining the design of an infrastructure service. Resource capacity estimates are result of the sizing process. Those (abstract) capacities must be further refined into actual resource quantities of certain resource types such as numbers of servers of a certain type. This is a second step after sizing, which also takes into account price and vendor preferences. Once these resource quantities of certain types have been determined, the blueprint for their connections must be developed and documented. This blueprint of sets of resources in combination with their connections (or wiring) is called a *Resource Topology*. Resource Topology is similar to wiring plans of electronic circuits, except that it is applied to resources as part of an infrastructure service design. It should be noted that Resource Topology is a design and represented as a model. It exists before actual resources in a data center are affected.

Resource Topology at the design stage has been a key part of research and is thus discussed in detail.

Another research area related to Resource Topologies concerned the generation of Resource Topologies based on formalized requirement specifications and known refinement rules automating the task of the solution architect of sizing infrastructure services and developing their blueprints.

7.1 Requirements for Plannning and Design

This section highlights a number of specific conditions and requirements that must be taken into account for the Planning and Design layer.

- Resource pools in enterprise data centers are large heterogeneous resource sets. For instance, different kinds of servers exist in an enterprise data center with different processors, architectures, resource proper-ties, operating systems, etc. Different kinds of storage resource exist. Multiple interconnect fabrics exist that are important resources and sometimes bottlenecks for connecting machines (via LAN) to machines and for connecting storage to machines (via SAN). Specialized devices such as firewalls or load balancers are important resources as well that need to be assigned and configured for enterprise applications.
- Enterprise resources provide substantially more capabilities that must be configured. Networks can be “programmed” as well as storage associations to machines. Different configurations can be applied to machines and devices depending on application requirements.
- Enterprise applications are composed of heterogeneous components that themselves are applications requiring resources. Enterprise application components requires different resource sets and different configurations (such as for the web tier, the application server tier, and the database backend).

- Applications have specific needs for resources in terms of resources they require at different times as well as in terms of specific configurations they assume on resources.
- Resources specifically need to be configured for enterprise application components. Examples are attaching specific disk images to servers, or creating and linking servers into specific networks.
- Resources need to be isolated when shared among different enterprise applications. Although this is a desirable goal, clusters typically do not provide application isolation due to lacking support in the infrastructure. A variety of techniques exist in enterprise data centers for isolating applications from each other. Virtualization is one important technique, ranging from using virtual LAN isolating IP address spaces to encapsulating applications in virtual machines. Application server containers are another example providing light-weight isolation among application components.
- Resource may need to be constructed and may be constructed in different ways for enterprise applications. For instance, an application component may require an "IA32 Linux PC". This resource may simply be taken from a pool as a physical resource, or it may be constructed using a Virtual Machine. Choices for creating virtual machines may exist (examples are VMWare's or Microsoft's Virtual PC virtual machines). Another option may be a user-level Linux partition. Choices for constructing resources must be explored.
- Resource constructions such as virtual machines are application processes themselves posing their own requirements onto resources. When a construction is chosen, the additional resource requirements of the constructions must be considered. Different costs can be associated with different construction choices in terms of resources constructions consume or licenses they require.
- Requested resource may not physically exist also for other reasons than constructions are being chosen. Resources may simply have not become part of the resource inventory in a data center yet, but are known to exist in future (e.g. when purchases are planned).
- Requests for Resources for enterprise environments are longer-term than traditional compute jobs (months/years vs. hours/days) leading to longer-term planning and allocation cycles for applications in enterprise data centers. Resource schedulers must accommodate long allocation cycles during which resource inventory may change.
- This also links to the consequence that resource requests may refer to resource inventory that does not currently exist, but is know to exist in future and hence can already be allocated. This case is important for enterprise environment facing continuous inventory replacement and turnover cycles.

7.2 Data Center Capacity Planning

The systems management discipline of capacity planning involves the planning of various kinds of resource capacities for an infrastructure [Schie02]. It is defined as

follows: capacity planning is a process to predict the types, quantities, and timing of resource capacities that are needed within an infrastructure to meet forecasted workloads.

Resource capacity involves four elements that are used in this definition:

- type of resource capacities required, such as servers, disk space, or bandwidth,
- the size or quantities of the resource in question,
- the timing of when the additional capacity is needed; and
- decisions about capacity that are based on forecasts of anticipated workload demands.

Data center capacity planning can be seen as the summation of all the capacity planning activities for all the applications hosted in the data center. It is particularly difficult, as pointed out in [Schie04]:

- Analysts and planners are too busy with day-to-day activities.
- Users are not interested in predicting future workloads.
- Users who are interested cannot forecast accurately.
- Capacity planners may be reluctant to use effective measuring tools.
- Corporate or IT directions may change from year to year.
- Planning is typically not part of an infrastructure culture.
- Managers sometimes confuse capacity management with capacity planning.

While the task of data center capacity planning has been addressed by practitioners and documented in guidelines [TQ04, Bla98] and best practices [HP08], the topic remains an important task in the Planning and Design Layer and has thus been addressed by research from different perspectives:

- the perspective of a data center [Gma07],
- the perspective of resource pools [Rol04] in data centers.

Likewise in IT management in general, capacity planning has been a fragmented process that has widely been performed independently of business considerations resulting from evolving and changing business processes conducted through IT systems as ultimate goal of IT. The need for more systematic and more integrated approaches to data center capacity planning is further emphasized by growing scale of next generation data centers.

The desire towards more integrated approaches to IT management also implies a more integrated approach to capacity planning and management, which has been addressed in more recent research, which is also highlighted in this section about data center capacity planning:

- develop automated performance engineering processes and tools for next generation data center management systems [Rol08],
- develop a systematic approach to derive IT configurations, including non-functional requirements, from business needs and processes [Gra08], and
- develop methods for integrated capacity and workload management for next generation data centers [Zhu08].

In the following sections, some research areas related to the Planning and Design Layer are discussed which had been investigated in more detail:

- Performance Engineering Processes for Data Centers [Rol08],
- Systematic Approach to Derive IT Configurations from Business Processes [Gra08], and
- Resource Topology Design [Gra05].

7.3 Performance Engineering for Data Centers

As data centers grow in resource scale (~capacity), human labor cannot scale proportionally in order to be cost effective. Developing a systematic process that provides the basis for automation has been the goal of this research effort called Performance Engineering Process [Rol08] assuming that in future IT infrastructure will be purchased as services from external and internal service providers. The Performance Engineering Process is for transaction oriented enterprise applications that supports infrastructure selection, sizing, and performance validation for customized service instances in hosted software environments. It enables the rapid deployment of a customized service instance while lowering performance related risks by automating the creation of a customized performance model and customized benchmark model. Case study results demonstrate the effectiveness of the approach for a TPC-W system.

For today's IT service providers, at best, service level agreements include service uptime guarantees. Performance is rarely addressed. However, performance will become more critical to support key business functions realized through IT services. For service provides arises the question how to accommodate performance needs from the applications side by sufficient and yet economical capacity of resources in their infrastructures.

7.3.1 Related Work

A range of topics contribute to automated performance engineering for services. These include benchmarking, workload generation, performance models, software performance engineering, automatic model generation, regression, as well as topics that relate to services including tenancy models and model-driven automation.

Benchmarking is a well accepted method for evaluating the behavior of software and hardware platforms [Grac96]. In general, the purpose of benchmarking is to rank the relative capacity, scalability, and cost/performance of alternative combinations of software and hardware. It does not attempt to directly predict performance behavior for any customized use of platforms. Instead, benchmarks aim to stress key features of platforms that are likely to be bottlenecks. Dujmovic describes benchmark design theory that models benchmarks using an algebraic space and minimizes the number of benchmark tests needed to provide maximum information [Duj99]. Dujmovic's seminal work also informally describes the concept of interpreting the results of a ratio of different benchmarks to better predict the behavior of a customized system but no formal method is given to compute the ratio.

Krishnaswamy and Scherson [Kri00] also model benchmarks as an algebraic space but also do not consider the problem of finding such a ratio. Krishnamurthy et al. [Kri06]

[Kri04] introduce SWAT which includes a method that automatically selects a subset of pre-existing user sessions from a session based e-commerce system, each with a particular URL mix, and computes a ratio of sessions to achieve specific workload characteristics. For example, the technique can reuse the existing sessions to simultaneously match a new URL mix and a particular session length distribution and to prepare a corresponding synthetic workload to be submitted to the system. They showed how such workload features impact the performance behavior of session based systems. APE exploits the ratio computation technique in this work to automatically compute a ratio of benchmarks that enables the creation of customized performance and benchmark models for a service instance. However, APE is more than the use of the ratio computation technique. APE is the overall approach for organizing and exploiting various kinds of model information to enable automated performance engineering.

Queuing Network Models (QNM) have been used as predictive models for enterprise computing systems since the early 1970's [Buz73], [Rei79]. Predictive models such as Layered Queuing Models (LQM) [Wood95], [Rol95] enhance queuing network models by taking layered software interactions into account. These include synchronous and asynchronous interactions between client, web, application logic, and database servers and have been shown to improve the accuracy of performance predictions for multi-tier environments [Kri08]. Tiwari et al. [Tiw06] report that layered queuing networks were more appropriate for modeling a J2EE application than a Petri-Net based approach [Tiw06] because they better addressed issues of scale. Balsamo et al. [Bals04] conclude that extended QNM-based approaches, such as layered queuing models, are the most appropriate modeling abstraction for multi-tiered software environments. The customized performance models considered here are LQMs.

Software Performance Engineering (SPE) offers a systematic approach that supports the design and sizing of enterprise software systems [Smith90]. Software control flow diagrams are used to describe execution paths through software modules that cause the use of system resources, e.g., CPU and memory. Modules that are expected to affect performance most are considered in greatest detail. The resulting information is used to create predictive performance models. By representing control flow and resource usage, the impact of software design or hardware changes on performance can be explored. Vetland [Vetl93] considers the challenge of systematically creating a library of resource demand models for a system's software components so that they can be integrated through SPE based methods to support design and sizing. Hrischuk et al. [Hri99] explores methods for automating the capture of control flow in software design environments and support automated model building. Petriu et al. [Petr07] consider the use of UML and other techniques to better enable software designers to directly exploit performance engineering concepts.

SPE related techniques all require methods to predict the resource demands of software components. However, predicting the resource demands of business objects in a software system is a challenging task. The two main reasons are: CPU and input-output usage are not typically measured with respect to software operations; and, per request demands are not deterministic. Measurement is difficult because today's application servers are complex. They are typically multi-threaded, execute on hosts that often have multiple CPUs, may execute on virtualized hosts, and have many layers of caching that frequently delay input-output activity. Furthermore, demands by requests for the same operation can

often cause very different resource demands depending on the state of the system. For this reason it is very difficult to create reusable resource demand models for software components that can be composed to reflect specific behaviors.

Research has looked towards statistical regression to estimate resource demands. Early work used regression techniques to estimate difficult to measure CPU demand overheads in a virtualized mainframe environment [Bard78]. Rolia, Vetland, and Sun used Ordinary Least Squares (OLS) [Rol98] and the Random Coefficients Method (RCM) for regression [Sun99] to estimate per-method resource demands for objects with many methods to support the creation of LQMs. RCM aims to overcome issues of non-determinism in demands. However, they found the general use of regression methods for the characterization of demands to be problematic because the assumptions of regression are violated, e.g., deterministic distribution for per-request demands. Furthermore, it can be difficult to decide how to group data for the regression. Stewart et al. considered Least Absolute Residual (LAR) regression techniques to predict demands for systems with different request types [Stew07]. A case study showed good results when characterization could be done under conditions where there was little contention for system resources, which they note is often the case for production environments. They also integrate an ad-hoc queuing formula into the regression formula to also predict response times. For the data they considered, they found that characterizing the demands for different types of requests enabled better utilization predictions than not distinguishing demands by request type, that using LAR worked better than OLS, and that their response time prediction technique behaved best for systems with low utilization levels. Zhang et al. apply a non-negative OLS regression technique to estimate the per-URL demands of a TPC-W system [Zhang07]. They used the demand estimates to create a simulation model and a QNM for the system. The simulation and QNM models resulted in similar prediction accuracy. Both predicted the throughput of emulated users often up to high system utilization. Mean response time estimates were not compared with measured values for the analytic model.

7.3.2 Approach

The Automated Performance Engineering process differs from the straightforward application of SPE and doesn't have to use a regression technique to predict demands. In contrast to SPE, the process focuses on system configuration and sizing through the selection of pre-existing business processes rather than on software design from scratch. It is assumed that each business process has a pre-existing control flow model that describes its expected execution of business process steps - based on the usage behavior of other service instances. If typical control flows are not representative of how a particular business will use business processes, they can be altered using SPE techniques. The resulting control flows and impact would automatically be taken into account.

The Automated Performance Engineering process helps to overcome the demand estimation problem by raising the abstraction level and predicting resource demands aggregated over many business objects rather than predicting per-business object resource demands. Business objects are rarely used in isolation so it can be difficult to characterize them separately and then predict their joint resource usage. This new approach to demand estimation is evaluated later.

Multi-tenancy hosts many service instances on one instance of a software platform. Isolated-tenancy creates a separate service platform for each service instance. A hybrid may share some portion of a platform such as a database across many service instances while maintaining isolated application servers. Multi-tenancy systems can reduce maintenance and management challenges for service providers, but it can be more difficult to ensure customer specific service levels. Isolated-tenancy systems provide for greatest opportunity for customization, performance flexibility and greatest security, but present greater maintenance challenges. Hybrid-tenancy approaches have features of both approaches. The Automated Performance Engineering process focuses on isolated-tenancy systems that operate in shared virtualized resource pools. However, the technique could also be adapted to reduce the performance related risks of the multi-tenancy and hybrid-tenancy paradigms as well.

Model-driven techniques have been considered by many researchers and exploited in real world environments (salesforce.com and SAP ByDesign [SAP08]). In general, the techniques capture information in models that can be used to automatically generate code, configuration information, or changes to configuration information. The general goal of model-driven approaches is to increase automation and reduce the human effort needed to support IT systems. Automation reduces costs, decreases the likelihood of errors when making changes to systems, and increases the rate at which systems are able to adapt based on business needs.

7.3.3 The Model Information Flow

This section describes a model-driven provisioning approach for hosting service instances. The approach implements a Model Information Flow (MIF) that provides the context for APE [Bel09]. The MIF presents a sequence of models and model transformations that support lifecycle management for information about a service instance. It captures requirements for a service instance, information about software platforms that may implement the instance, alterative feasible infrastructure designs, and information about a shared environment for hosting service instances. Model transformations support the manipulation of information about a service instance as it traverses its lifecycle. This section describes pertinent models and transformations that explain the following.

- How to choose an infrastructure design alternative for a service instance and estimate its number of resource instances so that it meets throughput requirements and response time goals.
- How a performance validation test can be deduced for a deployed service instance.

The MIF's lifecycle models include general, custom, unbound, grounded, bound, and deployed models. These models correspond to lifecycle stages for service instance information. Figure 10 illustrates the MIF. Model information is propagated from left to right as a service instance evolves towards deployment. Model transformations are used to support this evolution [Bel07]. In general, each model can be expressed using different formalisms, such as BPMN, CIM, or EMF.

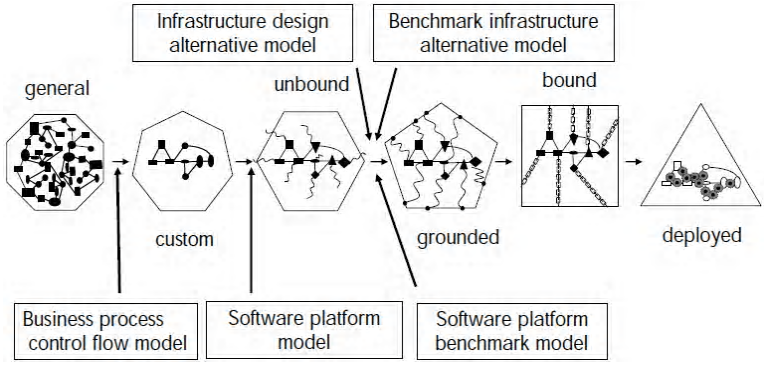


Figure 10: The Model Information Flow (MIF)

The transformations translate between formalisms as well as providing value added functions, e.g., design alternative selection. Information about a service instance may also move from right to left to support on-going management and maintenance. The lifecycle models are augmented by additional platform models that capture information about expected customer usage, and configuration and performance information about the vendor specific software and infrastructure platforms that ultimately realize the service instance. The focus is on the models and transformations that explain the APE approach to automated performance engineering for transaction oriented enterprise applications and services.

The *general model* describes business processes that can be deployed in an automated manner. It is a collection of business process models. Each business process model may have many variants and configuration alternatives that are captured in its model. Each variant has business process steps that must be realized by a software platform. Sales and distribution is an example of a business process. It may have business process variants that deal with orders, with orders for preferred customers, and with returns. Its business process steps may require information about items, may update an order, and may cause an order to ship.

The *custom model* includes only those business process variants needed by a particular service instance for a business. Additional business specific information is included in the customized model that expresses non-functional requirements such as each chosen variant's expected throughput and mean interactive response time goal.

The *unbound model* elaborates further on how software platforms implement the chosen business process variants. This model relates the variants to software platform business objects and technologies needed to implement the variants, e.g., Web servers, application logic servers, and database servers.

The *grounded model* is a design for the system. It relates the unbound model to a particular infrastructure configuration alternative from a catalog of feasible alternatives that can be automatically deployed to the shared environment. The grounded model includes estimates for the number of resource instances needed to support non-functional requirements. In includes information that enables the deployment of the service instance.

The *bound model* captures the assignment of real resource instances from a shared resource environment to the service instance. Bound resource instances are configured with software needed to participate in the software platform and management software needed to support ongoing management.

The *deployed model* describes resource instances participating in an operational service instance. Finally, a service for a business may have several deployed instances operating in parallel. Different instances may correspond to development, testing, and production environments. For example, a deployed service instance may be used for a performance validation test then discarded. Platform models augment the lifecycle models with vendor specific platform information. Examples of platform models are the business process control flow model; the software platform model; the software platform benchmark model; the infrastructure design alternative model; and the benchmark-infrastructure-alternative model. They directly support APE.

The *business process control flow model* describes the expected execution paths of customers through business process steps. For example, a control flow model may express loops to indicate that a step is executed multiple times and branches to indicate different alternatives for execution. The control flow models have estimates for loop counts and branch probabilities that are based on typical customer usage. Each business process variant has as at least one control flow model. Multiple control flow models may reflect the differing usage of a business process variant by different industries such as manufacturing or utilities.

The *application packaging model* expresses how a particular software platform implements business process variants from the general model using business objects and how the business objects relate to application servers. For example, a business process step that requires information about an item needs to access a business object for items. The software platform vendor also needs to estimate the number of visits to each business object by each process variant that it implements. The number of visits must correspond to the business process control flow models for the typical use cases of the variant. Finally, the aggregate business object usage for the software platform may require a particular subset of application server and database technologies. Such information is known by software platform vendors and is also an input to this approach.

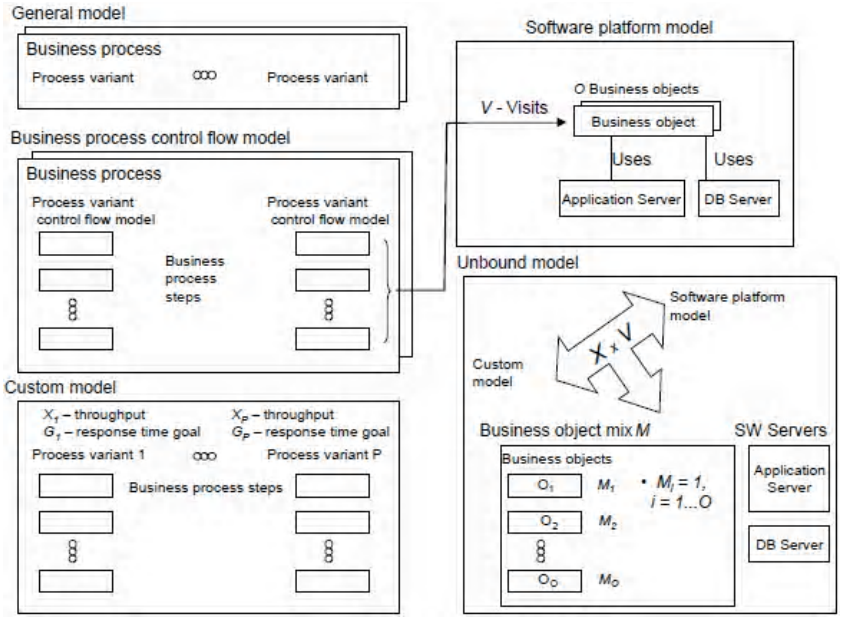


Figure 11 Models contributing to business object mix M.

Figure 11 Models contributing to business object mix M. illustrates the relationship between the general, business process control flow, custom, software platform, and unbound models. It shows how information from the general model, business process control flow, custom model and software platform model are used to compute a desired business object mix M for a customized service instance. The custom model includes a subset of process variants from the general model with specific control flows from the business process control flow model. Each of the variants is annotated with a required throughput, e.g., number of sales orders per hour, and a mean response time goal for interactive response times. Each business process variant causes visits to one or more business objects from the packaging model. The product of throughput and visits enables the computation of M. Finally, the identity of the business objects that are used also determines which of the software platform's software servers are needed in an infrastructure design alternative.

Benchmarks for a software platform help to automate the creation of customized performance models and customized benchmark models. The software platform benchmark model includes many benchmarks for a software platform. The benchmarks are chosen to provide coverage over the platform's business objects. Each benchmark is fully automatable in its execution and can be run on many different infrastructure alternatives. Each benchmark exercises a small number of objects in a manner typical for the platform. Together, the benchmarks exercise all the objects of the platform. Each benchmark aims to exercise an infrastructure to achieve the highest throughput while certain response time expectations are satisfied. Further details about benchmark design are given in the following section.

The *infrastructure design alternative model* expresses different ways in which a software platform can be realized in the shared resource environment. For example, one infrastructure alternative may have all software platform technologies executing entirely on one host with a specific capacity. This is referred to as a centralized design alternative. A distributed design alternative may use many hosts to realize a more scalable multi-tier system with each instance of a web, application, and database server running in a separate host. Furthermore, some alternatives may use hosts that are implemented as virtual machines. Finally, each infrastructure design alternative has a predictive performance model that is used to predict the behavior of the infrastructure when operating with different numbers of resource instances. Figure 12 illustrates the components of the infrastructure design alternative model. The figure illustrates some of the details considered in a model, e.g., logical and physical resources, resource capacities and networking relationships. The performance model reflects the behavior of the resources and their relationships.

The creation of infrastructure design alternatives is the role of the service provider. The service provider must create and test a number of alternatives that are appropriate for the hardware platform and that satisfy the needs of various customers for throughput and responsiveness. Once the alternatives are defined, they can be re-used and customized for many service instances.

The *benchmark-infrastructure-alternative model* acts a repository of reusable performance information for APE. All benchmarks from an application platform benchmark model are run against each infrastructure design alternative. The results of

each run include measured resource demands that are used as parameters for the corresponding predictive model. This process is automated with benchmarks being executed during non-peak periods for the shared resource environment. As new infrastructure design alternatives and different resource types and/or capacities are introduced, more benchmark runs are needed to keep the repository up to date.

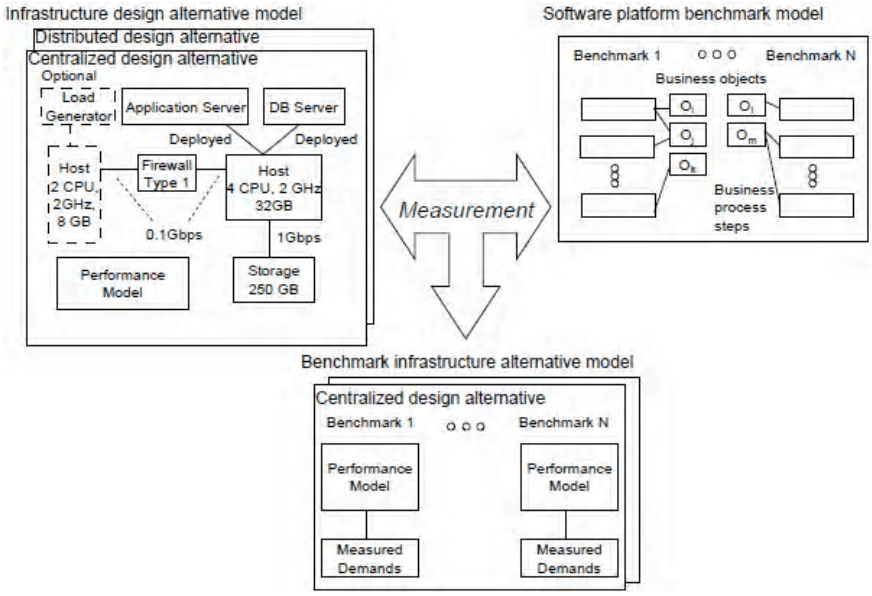


Figure 12: Benchmark performance models for infrastructure alternatives.

Figure 12 illustrates the relationship between the infrastructure design alternative model, software platform benchmark model and the benchmark infrastructure alternative model.

7.3.4 Case Studies

The case study has the following three objectives: 1.) demonstrate the effectiveness of computations for the workload integration ratio R; 2.) demonstrate the accuracy of APE's demand prediction approach and compare with regression; and 3.) demonstrate the effectiveness of the performance models by validating with respect to benchmark measurements.

For the case study, measurements from two different TPC-W [TCP] systems were used that were collected for unrelated studies [Kri06], [Zhang07]. Measurement data was gathered for these systems for other purposes but is reused to demonstrate and validate the concepts.

The first TPC-W system was deployed at Carleton University in Ottawa, Canada [Kri04], [Kri06]. The second system was deployed at HP Labs in Palo Alto, USA [Zhang07]. These systems are referred to as C-TPC-W and H-TPC-W, respectively. The results presented for these systems are not intended to be compliant TPC-W benchmark runs. The TPC-W bookstore system merely serves as an example system for the study. Detailed descriptions for the systems are available in their given references.

For each of the systems a LQM was created as a performance model. Performance estimates for the models are found using the Method of Layers [Rol95]. However, the TPC-W systems are complex session based systems. These systems have bursty request

behavior that is not well addressed using straightforward QNM or LQM-based technologies. As a result, for performance evaluation, the LQMs are combined with a population distribution estimation technique, conceptually related to the hybrid Markov Chain-QNM technique. The technique used is called the Weighted Average Method (WAM) [Kri08]. It improves the accuracy of performance predictions by taking into account the impact of bursts of competition for resources. Such bursts are typical for these session based systems. A more detailed description of WAM is beyond the scope of this discussion.

1) C-TPC-W Experimental Setup

The C-TPC-W experimental setup consists of a client node, a Web and application server node and a database (DB) node connected together by a non-blocking Fast Ethernet switch. The switch provides dedicated 100 Mbps connectivity to each node. The client node is dedicated exclusively to an httperf Web request generator that submits the workloads to the system and records request response time measures. The Web/application server node executes the Web and application server. It implements the TPC-W application's business logic and communicates with the TPC-W DB. The database node executes the DB server which manages the TPC-W DB. Finally, a performance monitoring utility is employed that collects a user-specifiable set of performance measures from both server nodes at regular specified sampling intervals.

The TPC-W application is deployed on Web, application, and DB servers that are part of a commercial off-the-shelf software product. The name of the product has been withheld due to a non-disclosure agreement with the vendor. The system is configured to not serve images. Image requests were not submitted in any of the experiments. The workloads that are considered are variants of the TPC-W workloads that include Hi-Mix, Med-Mix and Low-Mix workloads [Kri06] with high, medium, and low resource demand variation, respectively.

The number of server processes and the threading levels are set in the system as follows. The number of Web server threads is 1000. This was much greater than the maximum number of concurrent connections encountered in the experiments. The number of application server processes is fixed at 16, an upper limit imposed by the application. The number of DB server threads for the DB server was set to the upper limit of 32. The primary performance metric of interest for the study is the user-perceived mean response time (Rmean) for the requests at the TPC-W system. This metric is of interest for system sizing, capacity planning, and service level management exercises. Response time is defined as the time between initiating a TCP connection for a HTTP request and receiving the last byte of the corresponding HTTP response. The measured response time is a good indicator of the delay suffered by the request at the TPC-W system because the network and the client workload generator nodes are not saturated for the examples considered.

Figure 13 shows the LQM for the C-TPC-W system. LQMs are extended QNMs that include information about logical resources such as threading levels for application servers and software request-reply relationships. The LQM for the TPC-W system includes a think time centre and hardware resources. The logical resources in the model are the client browsers, Web server threads, application server threads and DB server threads. Threading and replication levels other than one are shown by placing a value

near the upper right hand side of an icon. For example, the Web/App server node has two CPUs. In this model, there are blocking requests between software resources and between software resources and hardware resources.

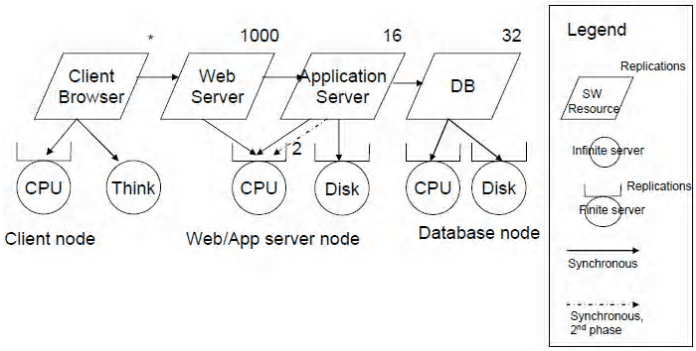


Figure 13: LQM for C-TPC-W System.

In Figure 13, one client browser corresponds to each concurrent session using the system. The number of client browsers is illustrated as a * and is managed by the WAM performance evaluation process. A customer using a client browser may visit its node's CPU or may think. A HTTP request causes a blocking call to the Web server. If a Web server thread is available then the request is accepted. The thread uses some CPU resource from the Web /application server node CPUs and then makes a request to the application server. If an application server thread is available then the request is accepted. The application server thread uses some CPU resource from the Web /application server node CPUs and then makes a request to the DB server. If a DB server thread is available then the request is accepted. The thread uses some CPU and disk resource from the database server node and releases the calling thread. The released calling thread from the application server can then complete its first phase of work and release the calling thread from the Web server.

From Figure 13 after finishing its first phase and releasing the calling thread from the Web server the application server thread continues on to a second phase of service. The second phase of service keeps the application server thread busy so that it cannot service another calling thread. However at the same time the calling thread from the Web Server that was released after the first phase of service can complete its work and release the calling thread from the client browser. This completes an HTTP request. The reasons for modeling the request-reply relationship of the application server in this manner are discussed shortly.

During an HTTP request, if a thread is not available when a server is called, the calling thread blocks until a thread becomes available. Once a thread completes its work it is available to serve another caller. Such threading can lead to software queuing delays in addition to any contention for hardware resources that are incurred by active threads. The numbers of threads used for each tier in the model reflect the actual application settings. To obtain resource demand values for the model, each measured run collects CPU utilization for the Web server threads, application server threads, and the DB server threads. CPU and disk utilizations were measured as well for the Web/application server node and the database server node, the elapsed time of the run, and the number of request completions. This enables us to compute the average resource demand per request for the

Web server threads, application server threads, DB server threads, and for the Web/application server node and database server node as a whole.

Finally, from measurement runs with one concurrent session was observed that the mean response times were often lower than the aggregate demand upon the hardware resources. This is an indication of two phases of processing at a server. This is reflected in the LQM by placing 25% of the application server thread demands in a second phase of service [Rol95], [Wood95]. This modeling choice was found to produce good model predictions.

2) H-TPC-W

H-TPC-W [Zhang07] was deployed on different software and hardware platforms than the C-TPC-W system. Furthermore, it also included image requests as part of its workload. Key features of the system are described here along with the performance model for this infrastructure alternative. The H-TPC-W system had two client nodes for emulated browsers, a Web/Application server running on a Web/App server node, and a DB server running on a Database node. This Web/Application server had a flexible number of server processes that varied with load. However, the actual number of server processes was not monitored during the measurement experiments. The DB server had a fixed number of processes that was large compared to the number of emulated browsers causing sessions. Measurement runs used the standard TPC-W workload generation method with parameters as defined by the TPC-W benchmark. Measured values included CPU and disk demands for each of the nodes and a response time value for each HTML request.

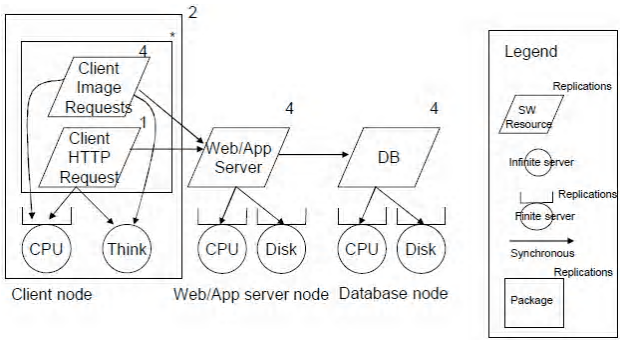


Figure 14: LQM for H-TPC-W System.

The LQM for the H-TPC-W system is shown in Figure 14. The model differs from the C-TPC-W system in several significant ways. Firstly, it did not require a second phase of processing for the Web/Application server. Second, the H-TPC-W system had image requests. To reflect the impact of image requests it was necessary to have two classes of customers in the LQM. The first class represented the HTML requests. The second class represented image requests that operated in parallel with the HTML requests. The emulated browsers permitted up to four image requests to be active in parallel for each active HTML request. Since the actual number of Web/Application server processes was not known, the numbers of Web/Application server processes and DB processes was adjusted to 4 and 4. This offered accurate mean response time and throughput predictions for the full suite of experiments. Finally, Figure 14 introduces the package concept into the LQM. A package groups modeled entities together in a manner that they can be replicated in unison. The dual client node is reflected in the model as a package with 2 replicates. Within each client node there are * active HTTP requests, each with 4 active

image requests. During the performance evaluation process, the value of * is varied by the WAM technique.

3) The Effectiveness of Computing a Workload Integration Ratio R

This section demonstrates the effectiveness of using a set of benchmarks to synthesize a business object mix. 100 sessions were used chosen randomly from TPC-W Browsing, Shopping, and Ordering measurement runs to act as a surrogate for software platform benchmarks of Figure 12. Each of the 100 TPC-W URL sessions corresponds to a benchmark. Each of the multiple URLs in a session, i.e., benchmark, corresponds to a business object of the software platform model of Figure 11. Two examples were considered.

The first example uses the workload matching technique with 100 benchmarks to synthesize the Browsing, Shopping, and Ordering mixes, respectively. The Browsing, Shopping, and Ordering URL mixes act as business object mixes. It was expected that the matching did very well at synthesizing the mixes since sessions used as the benchmarks were obtained from a workload generator that created sessions that corresponded to the specifications for these mixes. The second example presents detailed results for these three cases and 7 additional, but more diverse, business object mixes. The second example shows that workload matching can synthesize mixes for complex service instance scenarios.

Figure 15 shows the desired and synthesized business object mixes that correspond to the TPC-W Browsing, Shopping, and Ordering mixes. As expected, the 100 benchmark sessions were able to match the desired mixes precisely.

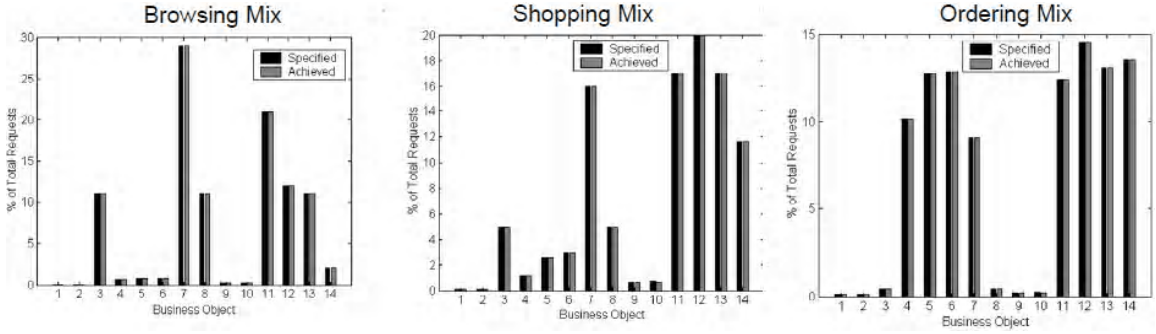


Figure 15: Using 100 benchmarks to synthesize TPC-W business object mixes.

Figure 16 shows 10 business object mix values for M and the corresponding mixes synthesized by workload matching, using the 100 benchmarks.

Each column in the data portion of the table corresponds to a business object, i.e., a URL. The numbers indicate desired and achieved business object mixes in percentages. The workload matching permits a specification of a tolerance, per business object, for matching each business object's mix. Even if an exact match cannot be found, close matches are typically possible as is shown in the table. To illustrate a diversity of mixes, note how the percentages associated with business object 4 goes from 0% in Mix1 through to 14% in Mix10. The percentages for business object 7 go from 31% in Mix1 down to 6% in Mix10. The percentages for other business objects also change. Mixes 2, 5, and 8, correspond to the standard Browsing, Shopping, and Ordering mixes defined by TPC-W, respectively. Mix11 and Mix12 are discussed in the next subsection.

Object		1	2	3	4	5	6	7	8	9	10	11	12	13	14
Mix 1	M	0	0	12	0	0	0.5	31	12	0	0	21	12	11	1
	M_{sch}	0	0	11.95	0	0	0.45	30.86	11.95	0	0	20.91	11.95	10.95	1
Mix 2	M	0.09	0.1	11	0.69	0.75	0.82	29	11	0.25	0.3	21	12	11	2
	M_{sch}	0.09	0.09	11.01	0.69	0.75	0.82	29.02	11.01	0.25	0.25	21.01	12.01	11.01	2
Mix 3	M	0.09	0.1	8	0.69	0.75	0.82	26	8	0.25	0.3	21	15	14	5
	M_{sch}	0.09	0.09	8	0.69	0.75	0.82	26.02	8	0.25	0.25	21.01	15.01	14.01	5
Mix 4	M	0.09	0.1	7	1.19	1.75	2.32	20	7	0.75	0.8	19	16	15	9
	M_{sch}	0.09	0.09	7	1.19	1.75	2.32	20.01	7	0.75	0.75	19.01	16.01	15.01	9.01
Mix 5	M	0.09	0.1	5	1.2	2.6	3	16	5	0.66	0.75	17	20	17	11.6
	M_{sch}	0.09	0.09	5.01	1.2	2.6	3	16.02	5.01	0.66	0.66	17.02	20.02	17.02	11.61
Mix 6	M	0.09	0.1	3	3.2	4.6	3	16	3	0.66	0.75	17	20	17	11.6
	M_{sch}	0.09	0.09	3.06	3.06	3.06	3.06	16.31	3.06	0.67	0.67	17.33	20.39	17.33	11.82
Mix 7	M	0.09	0.1	1	6.2	7.6	8	14	1	0.66	0.75	15	18	15	12.6
	M_{sch}	0.09	0.09	1	6.21	7.61	7.95	14.02	1	0.66	0.66	15.03	18.03	15.03	12.62
Mix 8	M	0.11	0.12	0.46	10.18	12.73	12.86	9.12	0.46	0.22	0.25	12.35	14.53	13.08	13.53
	M_{sch}	0.11	0.11	0.46	10.18	12.74	12.87	9.12	0.46	0.22	0.22	12.35	14.54	13.09	13.54
Mix 9	M	0.11	0.12	0.46	12.18	14.73	13.86	7.12	0.46	0.22	0.25	9.35	14.53	13.08	13.53
	M_{sch}	0.11	0.11	0.47	12.37	13.75	13.75	7.23	0.47	0.22	0.22	9.5	14.76	13.29	13.75
Mix 10	M	0	0	0	14	16	16	6	0	0	0	8	14	11	15
	M_{sch}	0	0	0	13.4	15.46	15.46	6.19	0	0	0	8.25	14.43	11.34	15.46
Mix 11	M	0.09	0.1	5	1.2	2.6	3	16	5	0.66	0.75	17	20	17	11.6
	M_{sch}	0.1	0.1	7.49	3.86	4.74	4.85	22.38	7.49	0.24	0.24	18.14	12.84	11.7	5.84
Mix 12	M	0.09	0.1	5	0	0	3	9	5	0.66	0.75	27.8	20	17	11.6
	M_{sch}	0.09	0.09	5.02	0.61	1.33	1.54	19.74	5.02	0.34	0.34	17.07	20.09	17.07	11.65

Figure 16: Synthesized mixes for 10 business object mixes.

7.3.5 Evaluation

The results for APE turned out to be better than regression for C-TPC-W and comparable for HTPC-W. Furthermore, the APE method is easier to apply within the automated approach. In particular, no decision is needed regarding the selection of the time duration for the rows of data. With regression, different capacity attributes sometimes benefit from different time durations. The regression techniques may sometimes require resource demand data be collected at short timescales. APE does not have this requirement. Furthermore, regression based techniques assume demands are deterministic and suffer from the problem of multi co-linearity as the number of variables grows. However, regression has an advantage that it can be used more readily to estimate demands for mixes than the APE approach which requires measurement runs for many mixes. Further study is needed to more definitively compare the accuracy of the two approaches.

7.3.6 Summary

The Automated Performance Engineering (APE) process supports infrastructure selection, sizing, and performance validation for customized service instances in hosted software environments. It enables the rapid deployment of a customized service instance while lowering performance related risks by automating the creation of a customized performance model and customized benchmark model. APE is described within the context of a model-driven approach for automating the deployment of service instances in a shared environment. The approach implements a model information flow that organizes the information needed for APE and other aspects of management. APE supports:

- choosing an appropriate infrastructure design from a set of alternatives for a service instance;
- estimating the numbers of resources that are needed for the alternative to satisfy throughput requirements and response time goals;
- creating a validation test that can be executed against a corresponding deployed service instance to verify its resource usage and performance characteristics.

Service providers can apply APE to estimate the initial resource needs of a service instance, predict the impact of changes to requirements for a service instance upon

resource needs, and to predict the impact of changes to a software or infrastructure platform on resource needs for all service instances.

APE depends on the ability to compute a workload integration ratio as a vector R, the ability to estimate the resource demands for a customized service instance, and on the ability of performance models to predict the behavior of complex multi-tier session based systems. Previously existing measurement results from two significantly different TPC-W systems were used to demonstrate the following: the effectiveness of the workload matching method for computing R; the effectiveness of the new method for predicting demands; and, the effectiveness of APE's customized performance models for predicting throughput and mean response time for customized service instances. The demand estimation technique performs well with respect to regression techniques for demand prediction and is easier to use. It is also less sensitive to the problem of multi-colinearity that exists when applying regression to benchmark runs [Stew07]. APE's performance models, along with WAM, outperformed other performance models that used the same data sets.

It can be concluded that the technologies needed to support APE are promising. Their effectiveness has been shown for two TPC-W systems. Further work is needed to better validate the new demand prediction approach.

7.4 Systematic Approach to Derive IT Configurations

This section extends the work described in the previous section about an automated performance engineering processes for capacity planning. This section extends this work by developing a systematic approach to derive IT configurations, including non-functional requirements such as performance and required capacity estimates, from business processes and other non-functional requirement specifications. It describes how the analytical work performed in the research is mapped into and automated process and the prototype implementation of tools, specifically for a case study related to SAP applications [Gra08].

When considering the traditional approach of how data centers are planned and how applications are being developed, built and integrated, three primary roles can be identified. A business consultant's role identifies and describes participants, processes, and non-functional requirements for qualities of service from the business' point of view. An application platform consultant's role is to customize application artifacts to realize the desired business processes. A solution architect's role then specifies and builds IT systems that support the processes with the desired qualities of service. This may include building new systems, implementing processes in existing or integrating across systems.

A major challenge is that information captured within the context of one role is often captured informally and communicated informally to those with other roles. As a result building solid and reliable enterprise applications remains to a large extent an art that relies on the knowledge and expertise of the consultants and architects. Published guidelines and established change management practices help mitigate risks [Sad06].

Goal of this research effort has been to develop a more integrated approach to the planning and design, configuration, and later deployment and management of data center applications.

Models described in the *Model Information Flow* (MIF – see previous section) are used to capture the specification information about processes, application platforms and IT infrastructures. Model transformations are used to automate many of the mundane steps currently performed by consultants. By automating the steps the aim is to reduce the time, costs, and risks associated with change. It is recognized that not all aspects of design and management can be automated.

The approach assumes several stages of descriptions in form of MIF models and the transformations between them. Models capture the various aspects of designing IT systems based on functional (business-logic) and non-functional (such as performance, capacity, sizing, but also security or availability) requirements. The MIF describes a chain of models ranging from the business process definition through various stages of refinement to a deployable and finally managed solution.

7.4.1 Related Work

A large body of work exists in the domain of business process design [Sche06], business process automation [Sche04] and business process management [Wes07]. Business process languages have widely been proposed and used, particularly in context of web services [Ley01]. Formal work on business processes has been done in the academic domain [Aal99]. More recently, collaborative business process environments have been proposed [Sad06].

On the other hand, IT automation and automated IT systems have become relevant trends in recent years in the IT industry. Initiatives such as Adaptive Infrastructure or Autonomic Computing reflect the situation enterprise customers face in growing complexity in IT systems along with increasing cost. Industrial research in this field reflects the need to support management of IT systems and ideally automate it [Sing05], [Gra07], [Gol04]. Integrated Service Management (ITSM) aims to streamline and standardize processes in overall IT management [ITSM] addressing not only the technical , but also organizational aspects in order to make IT management more efficient in enterprises.

However, although both worlds of business processes and IT systems are closely linked today in enterprises, they still remain widely disconnected conceptually organizationally and technically. Different groups of people with different background and skills work in the different domains.

The resulting fracture between the business layer and the IT layer already causes substantial pain today, and it is projected to become even more painful in future as the interactions between businesses continue to evolve. Only few intersection points have emerged between the domain of business and enterprise applications and IT systems such as business process monitoring or business-driven management [Bart04a].

7.4.2 Supplementing Business Processes with Non-Functional Requirements

A business process specification typically describes the functional requirements for a process. This is not sufficient for deriving an infrastructure configuration. In addition, non-functional requirements must supplement to the process definition such as information about performance, security, availability.

To narrow the problem space, the research focuses on performance aspects as part of the non-functional requirements. This section explains how a business process definition from the Customized Process Model is supplemented with performance requirements, which then allow to us to select, evaluate and parameterize a valid IT system configuration design, which is called the Grounded Model. Formalization through models enables the exploration of the design space for the Grounded model using techniques such as Layered Queuing Models (LQM) [Herz01], policy-based design [Gra05] and genetic algorithms [Vos99].

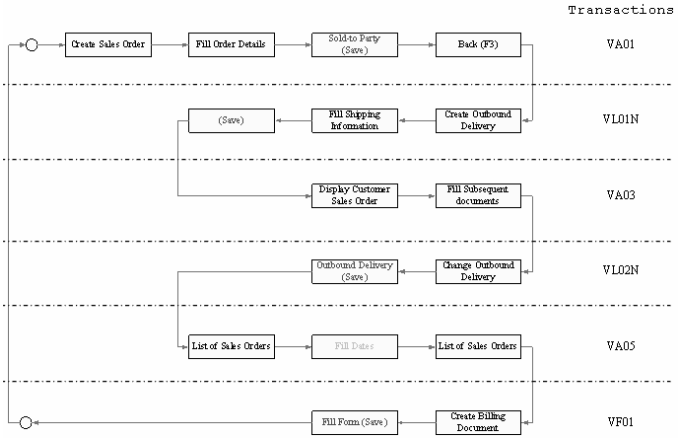


Figure 17: Example of Sales and Distribution (SD) process from SAP.

Figure 17 shows a simple business process (SD – "Sales and Distribution") which is used to benchmark traditional SAP systems. The process consists of 16 consecutive steps. Most steps are actions a user performs via the SAP user interface. These actions cause SAP transactions that are executed by dialog worker operating system processes (Dialog_WPs) within SAP Application Servers. The SAP transactions are application platform components that implement the business process.

Figure 18 shows a schematic view of the SD process as Customized Process Model in the Model Information Flow. The figure shows a UML fragment on the left introducing three classes (AI_Serivce, AI_BusinessProcess, AI_BPStep) and the associations between them. This definition is part of the schema definition of the Customized Process Model (see [Bel07] for the details of this model). The part on the right shows the model of the SD process in a business process editor (in a simplified form).

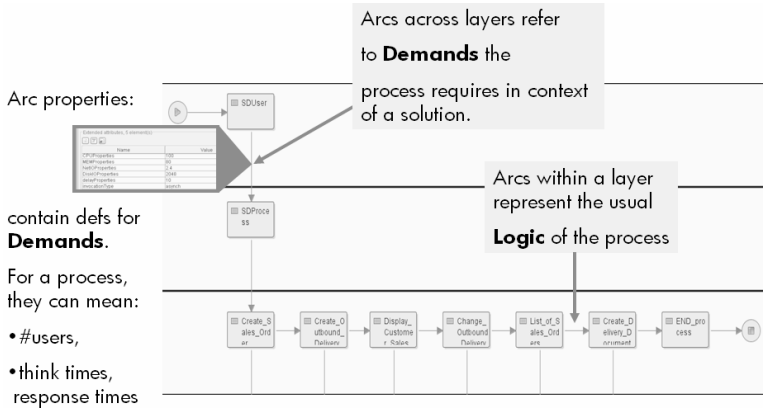


Figure 18: Supplementing demands for the SD process.

Figure 18 shows two kinds of relationships that are occurring in the process definition:

- *invocations* – such as SDUser invokes a process; arrows crossing layers represent invocation relationships, and
- *process logic* – such as CreateSalesOrder is followed by a business process step CreateOutboundDelivery; the arrows within a layer represent relationships between process steps.

The business consultant's view only shows the process logic relationships, not the invocation relationships. However, internally, the process definition is expanded to also reflect invocations based on knowledge about the external relationships of a business process. In a simple case, there is at least one entity (the node in the top layer) that will invoke the process, and there is at least one entity (the node in the second layer) that marks the entry point(s) into the process (the nodes in the third layer).

The demand information attached to the SDUser to SDProcess relationship is shown as the large arrow box in Figure 18. It contains few attributes expressing the desired number of users, mean think time, and required mean dialog response time that is expected for the process. Furthermore, the sequencing information for the business process steps is permitted to include loops and branches. For these expected values for loop counts and branching probabilities must also be specified. Together, requirements for users for each role along with the expected values for looping and branching gives the ratio of invocation for the business process steps. These values are entered into an editor by the business consultant as part of the overall requirement gathering process for the Customized Process Model.

It is important to note that non-process related information is not directly visible in the business consultant's view. Information that includes the relationship between business process steps and application platform components is added to the Unbound Model and attached internally to the customized process definition. This information comes from application platform vendors and may be manipulated by application platform consultants.

7.4.3 Component Performance Models: Capturing Component Demands

Component Performance Models are used to characterize detailed invocations among application platform components and demands upon infrastructure. Measurement based methods are used to characterize the demands of application platform components on the resource types supported in resource pools. Thus, different choices for resource types yield different demand values. Characterizations of resource usage are stored in the component performance model repository so that they can be reused. The values are then integrated to create customized process model specific application performance models for the design.

7.4.4 Design Templates: Describing Infrastructure Capabilities

Design templates are models that describe enterprise IT system variants that can be deployed in an automated manner. As examples, there may be a centralized design that deploys all application platform components and required application and database servers to a single operating system image. A distributed design would have application servers and a database residing in different operating system images. The templates are

designed to emphasize support for specific non-functional requirements. For example, a template may include firewalls than another to support a higher level of security.

The templates specify legal ranges for performance related configuration values such as the number of application servers, number of dialog work processes per server, and the concurrency level supported by the database. The choice of values depends on the system's non-functional performance requirements, its use of the application platform components, and the capacity of the resources used to support the system. The specification of these values plus the application platform information from the Unbound Model completes a Grounded Model. This is sufficient information to automatically proceed to Bound and Deployed Models.

7.4.5 Automated Evaluation of Configurations

In order to perform an evaluation of a configuration design, the three sources of information must be fused:

- *Customized Process Model* (business process steps and relationships with application components),
- *Component Performance Model* (application component demands on underlying system),
- *Design Templates* (IT system design including specific resource types).

Figure 19 shows the process of evaluating configuration designs based on the models.

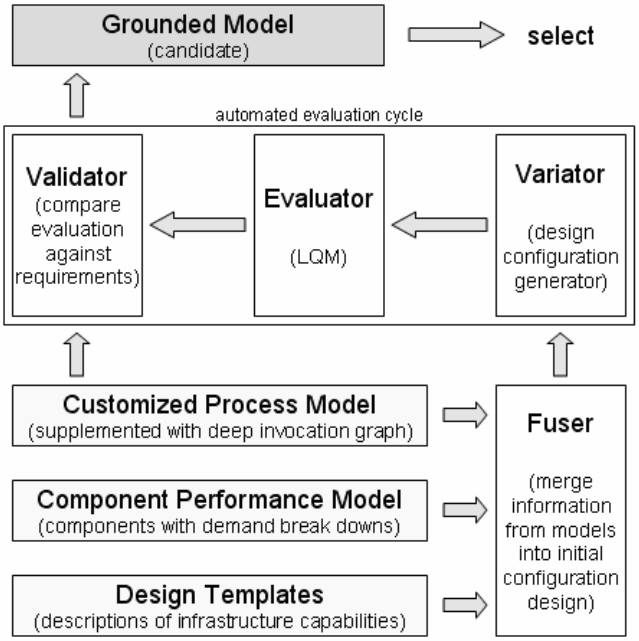


Figure 19: Automation process of deriving grounded models.

The *Fuser* reads the information from the three sources and merges it into an initial configuration design.

The *Variator* then consumes this initial configuration design and triggers the evaluation of the design. The Varitator will also identify possible permutations of configuration choices altering the initial configuration design and feed them into evaluation as well.

The *Evaluator* evaluates a configuration design. A Layered Queuing Model (LQM)-based tool [Rol95] named the Method of Layers (MOL) tool is used for that purpose. The result of the evaluation is list of latencies expected at the different components, e.g. at the database. Those latencies then can be compared against requirements leading to the exclusion of the evaluated design.

The *Validator* then can compare those latencies against requirements from the Customized Process Model.

If all the requirements are met by a design, LQM also provides information about initial numbers of instances of components such as numbers of application server instances needed to support the design. Those numbers then are supplemented into the evaluated design template turning it into a candidate for a Grounded Model which potentially could be deployed when it is selected. Initially, Grounded Models were considered produced by this method as a proposal to solution architects enabling them to make better founded decisions about infrastructure configuration choices. However, in future these Grounded Models may flow directly into deployment systems where their actuation can be scheduled.

Since the Model Information Flow is build on formalized models, interactions between its components can be automated. The automated evaluation of configuration designs is an example of that. The design space of the current research prototype is still limited. However, potentially large design spaces may exist in future which can be explored and automatically evaluated by this method.

7.4.6 Evaluation

As a case study, the evaluation of a SAP system was considered for a centralized system design. The table in Figure 20 shows several specifications for non-functional performance requirements along with corresponding performance configuration values. In all cases the mean customer think time was 10 seconds per dialog step as is typical for the SD benchmark.

Num User	Dialog Resp. Time Requ.	Num of Dialog WP	Num of Update WP	Num of Enqueue WP	Num of DB Process
1000	2000	8	2	3	12
1775	2000	83	3	16	102
1775	2500	55	2	4	61

Figure 20: LQM results.

The table in Figure 20 shows for 1000 SD users and a mean dialog response time requirement of 2000 milliseconds the enterprise IT system under study required 8 Dialog Work processes, 2 Update Work Processes, and 3 Enqueue Work Processes. The database had to be able to support up to 12 concurrent transactions. The considered resource type was able to support up to 1775 users with a mean dialog response time limit of 2000 milliseconds. For that scenario, significantly more Work Processes and concurrency at the Database was required. This suggests that the resource type must have significantly more memory (e.g. nearly 10x) than for the 1000 user scenario. By reducing the mean dialog response time requirement to 2500 milliseconds the number of Worker Processes and hence memory requirements drops considerably.

For a given set of non-functional requirements the LQMs enable us to report on how much capacity is needed for each design template alternative. The template that best satisfies non-functional requirements with the lowest requirement for capacity is recommended to the IT solution architect.

The IT solution architect interacts with the LQM evaluation tools though an Excel spread sheet. The input into the LQM spreadsheet is automatically filled in using the process shown in Figure 12. Figure 21 shows the spreadsheet that is presented to the IT solution architect.

	A	B	C	D	E	F
1	*To Invoke Optimizer -> Tools/Macro/Macro/Do_optimize*					
2	**Goal**					
3	NumberOfSDUsers	**1000**				
4	SDUserDialogMeanResponseTimeMs	**2000**				
5						
6						
7	**Factor**	**TempSpace**	**LowValue**	**HighValue**	**RecommendedValue**	
8	NumAppServer	1	1	1	1	
9	NumDialogProcessesPerAppServer	83	2	500	83	
10	NumUpdateProcessesPerAppServer	3	1	500	3	
11	NumEnqueueProcessesInCI	16	2	500	16	
12	NumDBThreads	102	2	500	102	

Figure 21: The LQM results as spread sheet.

Interaction with the evaluation results also allows to change parameters such as number of users for which a configuration is sized and re-running the evaluation. Designs can thus be changed and re-evaluated at a much faster pace then implementing them as test systems and exploring configuration choices in a real system.

7.4.7 Summary

The work presented in this section aimed at an intersection point between the business process domain and the domain of IT configuration design by establishing the linkage between both domains in form of the Model Information Flow.

The goal is to derive IT configurations from business configurations. In order to enable this function, additional information must be supplemented in form of the Model Information Flow. This information formalizes knowledge IT solution architects use to properly plan, design and size and implement enterprise applications. Information includes the structure of application stacks and performance characteristics of application and infrastructure components. All this information is then fused together using transformations.

A particular evaluation technique LQM has been presented within the context of the Model Information Flow that allows computing configuration parameters such as numbers of work processes to meet non-functional performance requirements provided in form of numbers of users and expected mean response time. A bounded space of design choices can automatically be evaluated. The final result is then used to configure the chosen infrastructure design.

7.5 Resource Topology Design

A Resource Topology [Gra05] is a model of a configured resource environment. Purpose of this model is to drive provisioning and deployment processes automatically rather than manually. A resource topology encompasses the entirety of resources (the resource set) with types and quantities of types as well as their connection relationships.

Resource Topologies must be designed and formally specified using a language and referred to in requests for resources for activation periods. A number of modeling frameworks and languages have been explored for representing Resource Topologies, including open formats such as RSL (Resource Specification Language [RSL]), DCML (Data Center Markup Language [DCML]), WSRF (Web-Services Resource Framework [WS-RF]) as well as proprietary formats such as the internal representation FML (Farm Markup Language) [FML] used in HP's Utility Data Center (UDC) [UDC].

Design tools are needed in order to develop Resource Topologies. Figure 22 shows an early tool which has been developed in HP Labs based on Microsoft's Visio environment. The Resource Topology Designer provided a library of symbols of typical data center resources such as servers, storage and network devices. Thos symbols are shown in the right panel. They could be drawn into the main design panel and placed into an arrangement. Properties of resources could be accessed and changed by right-clicking. In addition, connections could be added representing network connections between machines, attachments of disks to machines or locations of virtual machines on physical machines. The meaning of a connection was derived from the devices at its endpoints. By right-click, attributes of connections could be defined as well.

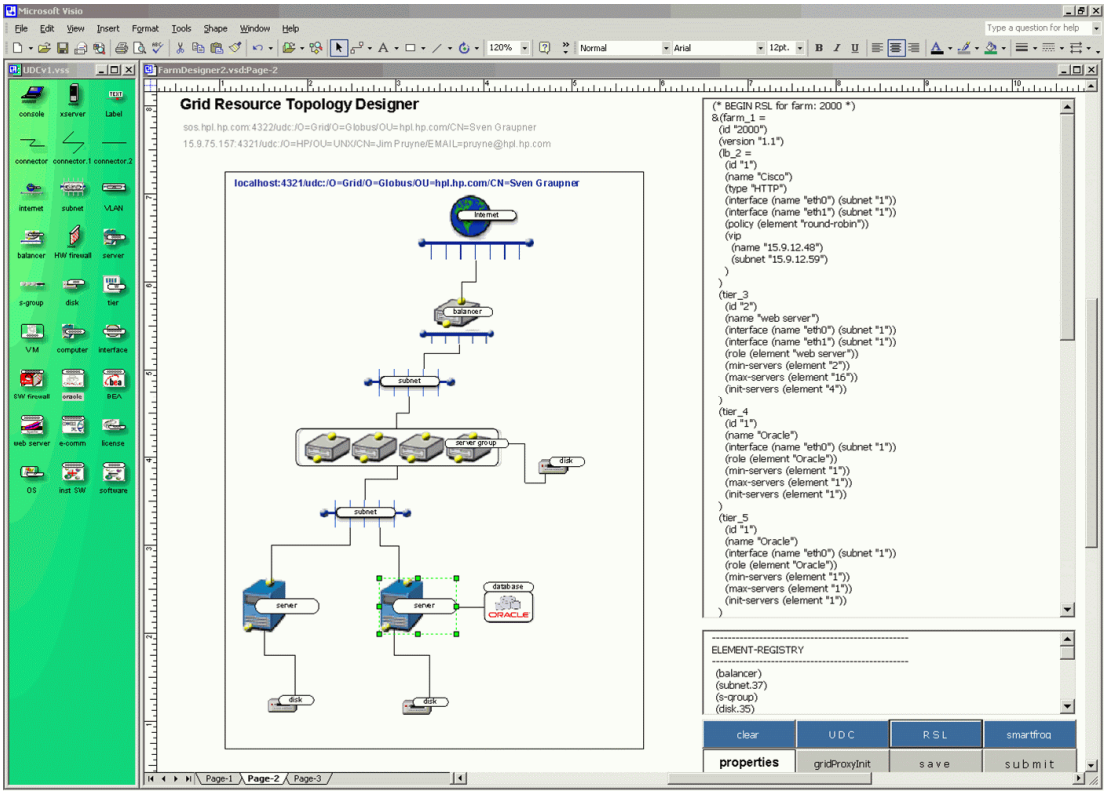


Figure 22: Resource Topology Designer prototype with Resource Topology.

Figure 22 shows a screen of an early version of the Resource Topology Designer which has been developed as part of research and demonstrated around 2004. It shows a resource topology and a formal specification in a language called RSL (Resource Specification Language, [RSL]).

Figure 23 shows the product-level realization of the Resource Topology Design prototype that has been developed earlier as part of the research.

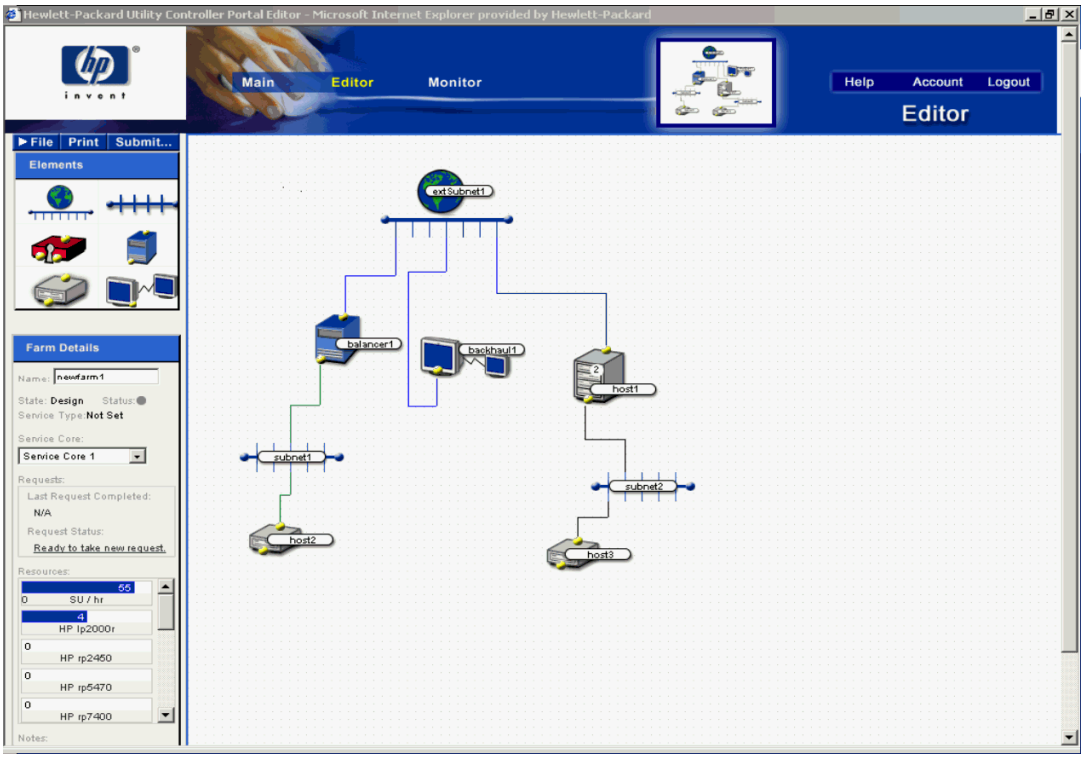


Figure 23: Product-level Resource Topology Designer used in the UDC.

Once the Resource Topology has been constructed in the main design panel, internal code analyzed the drawing and constructed a formal representation of the design, which can be seen in the panel on the right using the Resource Specification Language.

It is important that not only physical resource items are represented in a Resource Topology, but also software items such as an image placed onto a disk or an application placed into an image. Configuration flows could later be derived from this information automating the deployment processes when the Resource Topology was instantiated in a data center.

Another important aspect is that resource symbols used in Resource Topologies represent quantities of concrete types of resources, but not specific instances. Resource Topologies as designs should be reusable and multi-instantiatable in the same data center and across data centers requiring late binding of resource types used in the Resource Topology to actual resource instances of same types in an actual instantiation.

The decoupling of resource types and resource instances is also an essential enabler for dealing with virtualized environments [Gra03].

The formal RSL specification of the Resource Topology then could be exchanged and stored in a library of reusable Resource Topology Designs. Solution architects could download fitting designs and refine and customize them for their particular cases.

The process of refining and specializing Resource Topology Designs is also referred to as *Grounding*. The term Grounding originated in Grid computing from the need to formulate resource availability on one side and resource demands on the other in higher terms than specific hardware properties. For example, an application written for the Intel architecture would run on a number of processors with different properties from different vendors. Rather than promoting and requesting specific resource properties, such as vendor, clock speed or cache sizes, more abstract properties would need to be promoted and requested such as compliance to the Intel architecture instruction set, leaving more choices for matches. The idea of Grounding was adopted and extended to other resource types than processors.

7.5.1 Design Cycle of a Resource Topology

At the end, designs could be submitted to a provisioning system in a particular data center where the binding to actual resource instances occurred.

The *binding process* included the allocation of resources (either full physical resource entities or portions on shared resources) for the intended lifetime of the application instance. Allocation is the process of reserving a full or partial resource for an application instance for a given period of time. Allocation is a planning process as well. It affects resource assignments in the future, but not current resource use or assignments. A granted resource allocation is a commitment of the allocation system in a data center to fulfill a request for resources at the agreed time. A Resource Topology design can only successfully be allocated in a data center when all its resources can be allocated. In order to make this determination, instances for all resource types must be found available for the desired time periods and reserved.

The second stage of binding is activated shortly before resource allocations become due. At this point, actual resources must be taken from the environment and configured with the parameters from the Resource Topology Design. It is only at this point that actual resources in a data center become affected.

The process of applying specific configurations from the Resource Topology Design to actual resources is called deployment.

The planning and design stages of a Resource Topology Design are comprised of a number of stages:

- *Resource Topology Designs* are created using design tools and store them in a library of reusable Resource Topologies. Resources used in the design have the nature of concrete types and quantities of those types. They are decoupled from later bindings into full or partial resources from the environment of a particular data center.
- A solution architect selects a Resource Topology fitting his case and refines and specializes it to the design of infrastructure services for a particular Application Service. This process is called *Grounding*.

- The specialized Resource Topology design is submitted for allocation into a particular data center where a two-stage process of *late binding* occurs, which consists of stages of allocation and assignment:
    - *Allocation* grants or denies the Resource Topology design. It grants the allocation only when it can determine that all resource types and quantities used in the design are present and available for the desired time periods. If the allocation is granted, the full or partial resource quantities are marked as used and allocated.
    - *Assignment* occurs shortly before resource allocations become due for use. At this point they need to be underpinned with actual full or partial resources. For this second stage of binding the inventory of currently available resource instances must be examined and choices must be made to identify the resource instances matching the granted allocations. Choices for making those selections can include placement decisions, such as placements of servers resources for particular use from the Resource Topology Design, e.g. for use as a database server, onto certain physical servers in a data center. A decision whether a server resource is rendered as a physical server or a virtual server also belongs to the scope of the assignment (discussed as resource construction later).
    - Information about the selected set of resource instances is then supplemented into the Resource Topology Design making it "complete" to drive the subsequent stage of deployment.
    - *Deployment* (from the resource perspective) is the process of applying configuration information from the Resource Topology Design to assigned resources in order to produce the desired resource properties.

7.5.2 Automated Resource Topology Lifecycle

Automating the Resource Topology Designs at Planning and Design Stages has been a goal or research. Figure 24 shows a workflow along which Resource Topology information flows. Once the resource topology design has been completed using a design tool, the “non-grounded” resource topology will be routed to a resource composition engine that performs resource grounding. After grounding, the final resource topology is either forwarded directly to a resource allocation service or to a broker for locating resources in data centers.

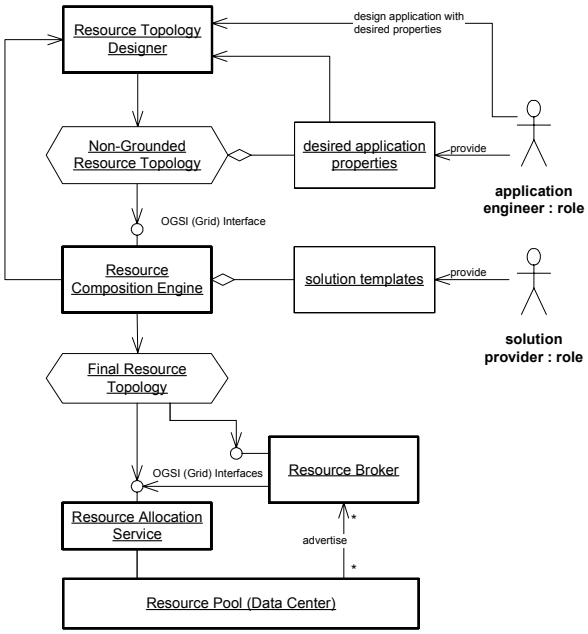


Figure 24: Automating the Resource Topology lifecycle.

The automation workflow shown in Figure 24 links to another research area addressed in this area: policy-based configuration.

7.6 Policy-based Configuration Generation

Goal of policy-based configuration is the ability to automatically generate grounded Resource Topology Designs from more abstract blue prints or templates. The technique is based on a policy engine (the Resource Composition Engine shown in Figure 24) which has been developed as a research prototype [Ram06]. Specifically the area of resource Grounding has been considered for policy-based configuration.

As mentioned, the term Grounding originated in Grid environments where requests for resources are specified in terms of quantities of actual compute nodes with certain properties: [CPU=IA32, MEM>2GB, DISK> 250GB, OS=Linux]. A Grid request could include 128 nodes of this type. Grid's early Resource Specification Language [RSL] allowed to describe resource requirements in those terms.

Resource requests in RSL can then be sent to a Grid Resource Allocation Manager (GRAM) representing a pool of resources for allocation. The GRAM evaluates the request and decides about acceptance. A variety of GRAM implementations exist as schedulers such as PBS [PBS] and Maui [Maui].

Co-allocation in a Grid allowed exchanging resource request among GRAM's (with or without broker) for identifying a resource pool in a Grid that would support the set of requested resources.

Other work extends match-making between resource requests and available resources using the semantic web framework [Ver01].

Described techniques in Grids are not sufficient for enterprise environments for following reasons:

- Heterogeneous sets of resources must be requested as entirety for hosting an enterprise application.
- Resource requirements requested by applications must be separated from realizations (and eventual constructions) of resources supporting requirements.
- Choices that exist for realizing each required resource must be evaluated in terms of conflicts that may exist to constructions of other resource, in terms of cost associated with constructions, and in terms of overall impact of constructions on the application system.
- Further requirements may need to be obeyed in terms of isolation and overall placement of resources in the data center.

7.6.1 Resource Properties for Policy-based Configuration

Policy-based configuration takes a request for an abstract resource into a request for grounded resources. A resource can be abstract in several ways:

1. *Abstraction and polymorphism:* A resource topology can be requested that can be realized in several different ways and which way to exactly realize it is left to the discretion of the grounding system. To understand this better, consider the analogy of abstract classes (classes that are defined as abstract) in object-oriented languages. A request for an instance of an abstract class can only be satisfied by creating an instance of one of its derived classes. The requestor does not necessarily care about which derived class is instantiated. An example of a polymorphic resource that could be grounded in two different ways is present in use case U1.1 below.
2. *Aggregation:* A resource topology can be requested that is aggregated from several other component resources. Instead of asking for each of the component resources, a requestor could simply ask for the aggregate resource. An example of this is in use case U1.2.
3. *Restriction:* A resource topology can be requested that satisfies certain constraints – for e.g., constraints on a resource's properties, on the methods it supports, or on the relationships it has with other resources. The requestor is not aware of the exact resource that satisfies all these constraints. An example of this is in use case U1.3.
4. *Combination of the above:* A request can be made for an aggregate resource that in turn contains polymorphic resources. In addition, the requestor could specify constraints that limit the choices of grounding. An operator could specify an additional set of constraints that further restrict the grounding choices.

Few use cases should illustrate the use of grounding. Grounding is the stage after a resource topology has been designed.

7.6.1.1 Polymorphic Resources

A requestor requests a firewall to be provisioned. The resource pool contains CISCO 6509 switches that can be configured as firewalls. The pool also contains servers that can

be turned into firewalls by installing Checkpoint software. In other words, the abstract firewall has to be grounded into one of the two types of firewalls by the grounding subsystem.

7.6.1.2 Aggregate Resources

A requestor requests a "three-tier farm". A three-tier farm is known both to the requestor and the operator as a group of servers arranged and connected in a special way as shown in the Resource Topologies in Figure 22 and Figure 23. The three-tier farm has three tiers – Web server tier, application server tier, and database tier. Each tier consists of a number of servers that are connected to one or more subnets. The Web server tier is connected to a subnet1 and subnet2. The application server tier is connected to subnet2 and subnet3. The database tier is just connected to subnet3. In addition, each server in the Web server tier should have Web server software (polymorphic) running on any operating system (polymorphic). Similarly, servers in the application server tier should have application server software and servers in the database tier should have database software installed on them.

7.6.1.3 Constrained Resources

A requestor requests a firewall as in Use case U1.1, but this time constrains the firewall to have the ability to support a certain traffic rate and a certain cost. Satisfying this constraint may require the grounding subsystem to make the choice of using Checkpoint software based firewall rather than CISCO switch. In addition, it may restrict the choices on which platform to use for the server and which operating system to install on it. This is because each platform and operating system has a certain cost associated with it. Also, not all operating systems can be installed on all platforms, which further constrains the possible choices. In this example, some constraints are intrinsic to resources (for e.g., which operating systems can be installed on which platforms), some are specified by the operator (for e.g., the costs associated with each type of firewall), while others are specified by requestors (for e.g., desired traffic rate and desired cost).

7.6.2 Resource Construction Model based on Constraints

When resources are combined to form other higher-level resources, a variety of rules need to be followed. For example, when operating systems are loaded on a host, it is necessary to validate that the processor architecture assumed in the operating system is indeed the architecture on the host. Similarly, when an application tier is composed from a group of servers, it may be necessary to ensure that all network interfaces are configured to be on the same subnet, or that the same version of the application is loaded on all machines in the tier. To ensure correct behavior of a reasonably complex application, several thousand such rules may be necessary if the construction of such applications is to be automated. This is further complicated by the fact that a large fraction of these rules are not inherent to the resources, but depend on preferences (policies) provided by the system operator or by the customer as part of the request itself.

In this section, a model for combining resources is proposed which allows specification of such rules in a distributed manner. By capturing the construction rules as part of the specification of resource types, and by formalizing how these rules are combined when

resources are composed from other resources providing a very flexible model for policy-based resource construction.

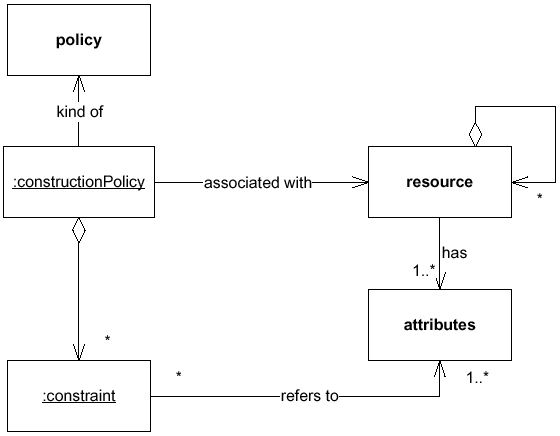


Figure 25: Conceptual model for resource construction.

Figure 25 shows the basic conceptual structure of the model. Each resource is defined in the model by a resource type definition. It is assumed that resources are described by attributes that are part of the resource type. These attributes reflect configuration or other parameters that are meaningful for resource construction. It is also assumed that resources can be composed from aggregates of other resources, and that new resource types can be constructed by combining other resource types.

Instances of construction policy are associated with each resource type. Construction policy instances contain constraints that are defined using the attributes present in the resource type definitions. When a resource is instantiated, the resource management system ensures that all constraints specified for that resource are satisfied. Because resource types can be derived from other resource types, this implies that all constraints for all composing resources are also satisfied. Because the resource manager creates the union of all (relevant) constraints when instantiating resources, it can also accommodate a variety of operator and user level policies during instantiation.

Construction policy is modeled as shown in Figure 26. Every instance of a construction policy is associated with an instance of one or more resource types. Construction policy is modeled as an aggregate of one or more constraints that are defined using one or more attributes in policy.

Policy attributes usually refer to attributes of the associated resource type, but may also be internal to the policy definition for convenience.

A policy applies to all the associated resource types. When the resource type is instantiated, the instantiated resource is defined to be in compliance with the construction policy if all policy constraints that refer to the attributes of the corresponding resource type are satisfied by the resource instance. Conversely, an instantiated resource is defined to be in violation of construction policy if any constraint that refers to an attribute of the corresponding resource type is violated by the resource instance. Note that it is possible for an instantiated resource to be in compliance, while the construction policy is in violation if the violation occurs as a result of a constraint that does not depend on any attribute of the resource. Additionally, if the constraint causing the violation refers to attributes in multiple resources, it may not be possible to uniquely determine which

resource is causing the policy violation. Unlike traditional [Slo01] models of policy, no action is defined in the construction policy model.

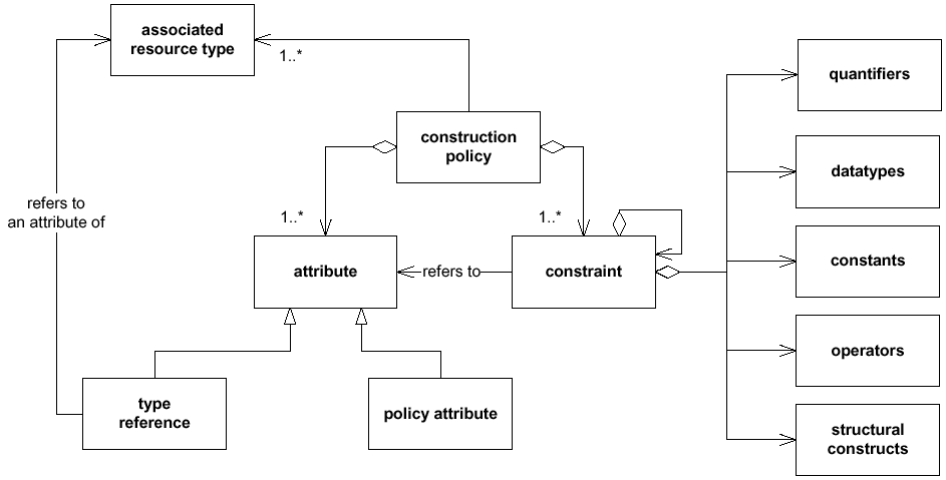


Figure 26: Elements of construction policy.

Constraints form the core of the policy specification and are defined using expressions that use policy attributes as variables. During the instantiation process, the attribute values from the corresponding resource instances are used to validate policy. Constraints contain first order predicates, arithmetic and logical operators, and other structural constructs defined below:

Data types. Data types may be imposed on attributes as constraints that have to be satisfied by the corresponding attribute, e.g. constraints can specify if a particular attribute should be a String, integer, float etc. This allows validation of data types when different underlying components are used to provide similar functionality.

Constants: Numeric or string constants may used in constraints for defining the values or thresholds for attribute values.

Quantifiers: Quantifiers are often used in constraints, e.g. ∀ (for all), ∃ (there exists), etc.

Operators: A number of operators can be used to combine attributes in defining constraints. These operators fall in the following categories:

- *Arithmetic operators* (+, -, *, /): These operators can be used for constructing arithmetic expressions on literals of the allowed data types.
- *Comparison operators* (<, >, <= , >=, ==, !=): Comparison operators can be used to compare other expressions, and result in a boolean value.
- *Boolean Operators* (&&, ||, ! (unary not)): Provide logical expressions in constraints.
- *Implication Operators* (==> (logical implication), <== (reverse implication), <=> (equivalence, or if-and-only-if)): These operators allow expression of dependencies between attributes, e.,g.
- (name == Solaris) ==> version \in {5.7,5.8};
- *instanceOf Operator:* The <: operator is used to denote "an instance of" relationship. This allows constraints to be created that enforce data types on

components or their attributes, e.g., ensure that component "server" is an instance of type Appserver<: AppServer;

- *Set Operator:* \in operator may be used to constrain values of an attribute to be always in a set.

Structural Constructs: Other structural constructs (e.g., let in, if then else etc.) are used mostly for syntactic convenience. These familiar programming constructs simplify the task of the constraint writer when complex constraints have to be expressed in policy.

Operationally, this model allows constraints to be specified in a distributed and hierarchical manner. The instantiation process is shown in Figure 27.

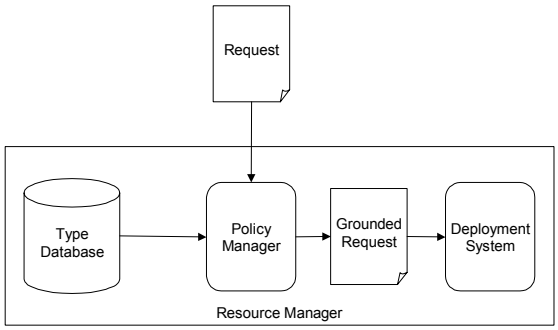


Figure 27: Resource construction process.

The resource manager contains a type database containing the resource type definitions (and the associated construction policies). A user submits a request to the resource manager for some resources. The request may contain additional policy constraints desired by the user (using constructs similar to the ones in the resource type database). The resource manager extracts the corresponding types from the database, and sends the resource request and the types to the policy engine [Sah04]. The policy engine treats the constraints and types requested by the user as a goal to be achieved. It treats the problem as a constraint satisfaction problem, and uses a constraint satisfaction engine [Hen89] to assign values to all attributes in the resource type definitions, such that all of the policy constraints are satisfied. The output of the constraint satisfaction engine is a request specification (a grounded request) where all attributes have been filled out. This grounded request is then handed to the deployment manager [SF04] for actual instantiation.

Note that because the policy engine assigns values to all the attributes such that all the constraints are satisfied, explicit condition-action pairs are not needed in construction policy. Thus, the model allows complex configurations to be built without requiring the user or the operator to pre-specify which combinations are valid and/or having to explicitly specify how such combinations can be achieved.

7.6.3 Examples of Applying Construction Policies

In this section, a number of examples is discussed to highlight how the model of construction policy can be used in practice.

7.6.4 Resource Composition

When composing higher-level resources from other resources (e.g., an e-commerce site from servers), a variety of resources need to be put together. However not all possible combinations are valid. Policies can be attached to component types to ensure that the resulting construction is valid. To illustrate this principle, an example is assumed to create a number of servers. A server is simply defined as a computer system with an operating system on it. In order to create servers, computers need to be selected and operating system images need to be installed on them. A Server resource entity is thus constructed out of an underlying Computer resource and an OperatingSystem resource. However, not all computer types may be available in the resource pool, and not all operating system images would work on a given computer. Thus constraints need to be defined that identify which computer and operating system image combinations are valid for constructing a Server. Figure 28 show one possible way of doing this.

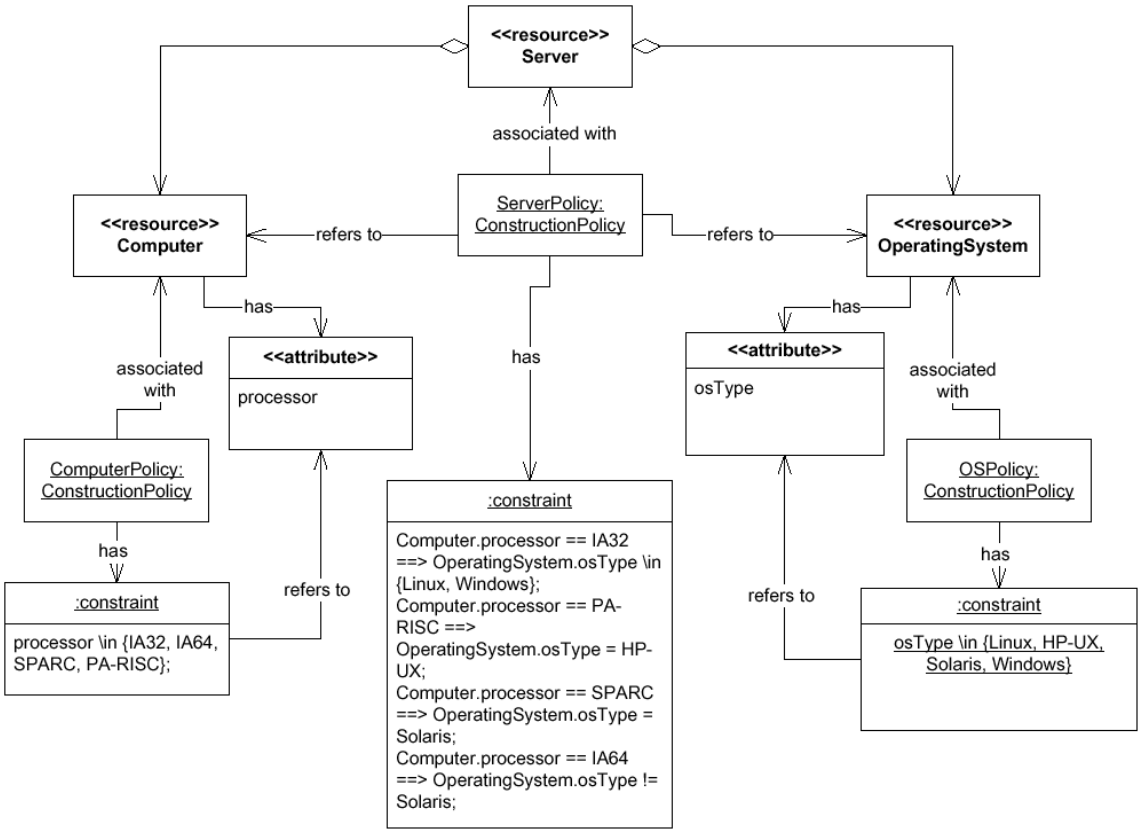


Figure 28: Example of a composition policy for a server resource.

In the Figure, a Server is a resource type that is composed from two other resource types: a Computer, and an OperatingSystem. A Computer has an attribute processor while an OperatingSystem has an attribute called osType. A policy associated with the Computer type states that the attribute processor can only take values in the set {IA32, IA64, SPARC, and PA-RISC}. Note that this constraint is specified by the system operator (perhaps because only these instances are available in the resource pool). Similarly, the construction policy associated with the OperatingSystem resource type states that the attribute osType can only take values in the set {Linux, HP-UX, Solaris, and Windows}. Again, the set is defined by the operator based on available operating systems within the

environment. When the Server resource type is created, the type definition includes policy constraints that specify which osType can exist with which processor In order to create a valid instance of Server.

Suppose a request specifies that it needs a resource of type Server with the additional constraint that Server.Computer.processor = PA-RISC. From the constraints specified as part of the Server, the policy system determines that the only valid value for OperatingSystem.osType = HP-UX, and automatically fills that value.

This example shows a number of aspects of constraint-based construction policies:

- By associating policy constraints with the individual component types, construction using those types can be controlled. By changing the allowed policy constraints, valid (or available) configurations can be maintained by the operators, and easily changed as needs change without extensive code or system-level modifications about how the new types are handled.
- Policy constraints for a resource type depend only on the attributes of that type, or the attributes of the underlying resource types. This simplifies hierarchical specification of types, because such dependencies can usually be localized in the type hierarchy. The designer of a new type only has to deal with the preceding types it is using and the corresponding constraints on them. The policy system automatically accounts for other dependencies that are created through transitional relationships.
- The policy system checks all constraints for validity when handling resource requests. Because it can locate constraints that are being violated, the request specification can be checked for "correctness" with respect to those constraints. Similarly, during the instantiation of new resource types, the policy system can aid the designer by validating that at least one valid instance of the new type can be created.
- The system can also fill in attribute values based on correctness of constraints. This means that the requestor has the freedom to only specify the attributes that are meaningful, and let the system fill in the gaps. This simplifies the requests.

The requestor can add additional constraints for the policy system as part of the request. Because the policy system forms the union of all constraints when constructing the system, request-specific policy can be easily incorporated during construction. Many additional constraints can be added to this simple example to account for items such as licenses, software versions, or other attributes such as memory and CPU speed. In addition, higher level resources can be constructed in a similar manner. For example, a Server may take the role of a webServer, an applicationServer, or a databaseServer if the corresponding images are installed on it. Furthermore, by adding topology constraints (e.g. web server tier precedes an app server tier which in turn precedes a data base tier), one can construct much more complex resource types like a three-tier architecture or a highly available server and treat them as higher-level resources. These complex resource types may then be instantiated and deployed.

7.6.5 Component Selection

Multiple components that offer the same base capability are often available in the resource pool. Examples may be different types of servers, firewalls, or network switches.

However, these components frequently differ in the capabilities (e.g., security, availability, throughput) offered by them. Such capabilities can be captured by the attributes of the components, and depending upon the capabilities desired by the requestor, the appropriate components can be selected to meet the users' requests.

In order to demonstrate component selection, four examples are discussed in detail to highlight how policies can be used to provide construction choices for components.

Examples presented in the following four subsections include:

- Capability-based component selection.
- Hardware and software partitions in a server.
- Multi-function and polymorphic resources.
- Class-of-service based resource selection.

7.6.5.1 Capability-based Component Selection

Suppose multiple types of switches are available in the resource pool. The switches offer different levels of security capabilities. For example, some switches may have Kerberos authentication or secure shell capabilities implemented, while others may not. Rather than forcing the user to understand all the details of the switches, the system could allow the user to simply specify that she wants a switch fabric that supports secure shell access, and automatically select the appropriate components to meet that request. This is shown in the example below, where three different kinds of Cisco Catalyst switches are available, and the security capabilities needed are specified in the request.

<<code>>

```
// this component captures various switch attributes. Most likely extended from CIM
Switch {
   manufacturer: String;
   switchFamily: String;
   model: any;
   security: SwitchSecurityCapability
   // other attributes
}
// this component captures security capabilities of switches
SwitchSecurityCapability {
   PortSecurity: boolean;
   TACACSauthentication: boolean;
   AdvancedLayer3and4ExtendedAccessList: boolean;
   DynamicACL: boolean;
   PolicybasedRouting: boolean;
   NetworkAddressTranslation: boolean;
   SNMPv3: boolean;
   secureshell: boolean;
}
// this component defines the capabilities of available Catalyst switches
CiscoCatalystSwitch extends Switch {
   enum availableModels {WSC3750G24TSE, WSC2950ST24LRE, WSC4912G};
   satisfy (manufacturer == "Cisco");
   satisfy (switchFamily == "Catalyst");
   satisfy model \in availableModels;
   // security capability support in the available models
```

```
    satisfy (model == WSC3750G24TSE) ==>
        (security.portSecurity == true && security.ACL == true &&
         security.kerberos == true &&
         security.TACASauthentication == true);
    satisfy (model == WSC2950ST24LRE) ==>
        (security.portSecurity == true && security.SNMPv3 == true &&
         security.ACL == true && security.TACASauthentication == true &&
         security.secureshell == true);
    satisfy (model == WSC4912G) ==>
        (security.portSecurity == true && security.SNMPv3 == true &&
         security.TACASauthentication == true &&
         security.secureshell == true);
}

// request: This example shows how the security policy in a request
// selects a switch for the implementation. "main" is a special component
// that defines the system requested
main {
    sw: Switch;
    // this constraint specifies the desirable security
    // capabilities needed for switch
    satisfy (sw.security.ACL == true && sw.security.secureShell == true);
}
```

Figure 29: Capability-based component selection.

The example has been written using the grammar specified in [Gra05] to show how the components (and the request) could be formulated. Again, note that:

1. In this example, if switches with other capabilities become available, the operator can simply add those types to the system with their corresponding capabilities.
2. Because the policy system finds attribute values that satisfy all constraints, an appropriate model of the switch is automatically selected.
3. If multiple switches satisfy the user's request, the policy system is free to choose (an arbitrary) switch from the list of switches that satisfy the request. Other policies (such as cost of the solution) can be added in the policies to further restrict how the selection takes place. Note however, that the constraint satisfaction engine does not solve an optimization problem. Thus it is currently not possible to ask for a "minimum cost" solution, although "cost < 500" is an appropriate constraint. Merging optimization and constraint satisfaction problems remains a subject of research.

Transient or virtual resources (e.g., virtual machines) are instantiated, allocated and removed just like other resource instances. However, unlike other resources, transient resources do not have underlying a-priori resource instances in the resource pool. Let us consider an example related to server partitioning. A Server not only can be allocated and used as a single resource; it may be partitioned for better resource utilization. Server partitions enable multiple applications to run on the same server while maintaining isolation among them, thus improving server utilization. A Server can support both hardware and software partitions. Multiple software partitions may be supported within a hardware partition. A hardware partition is termed an nPartition (nPar) and a software partition is termed a Virtual Partition (vPar) as shown in Figure 30.

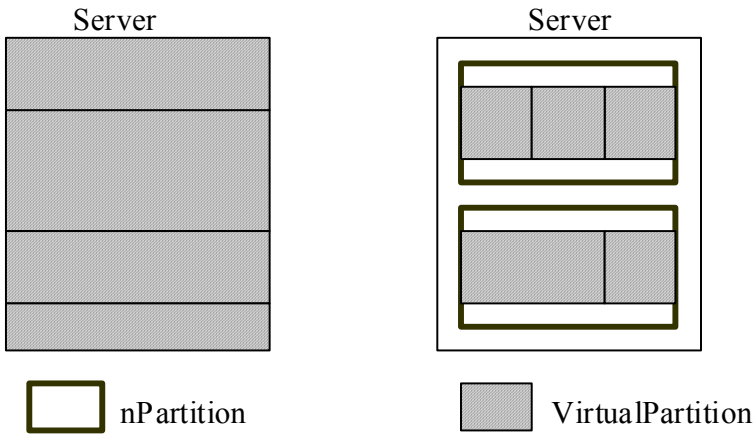


Figure 30: Hardware and software partition overlays for a server resource.

Usually, there are restrictions on the number of nPars and vPars that may be created on top of different types of servers. For example, an nPar may be restricted to have 2, 4, 8 or 16 CPUs in it.

7.6.5.2 Hardware and Software Partitions in a Server

The following example shows how such restrictions can be described as policy constraints in the types defining virtual servers.

<<code>>

```
// A virtual partition
VirtualPartition extends Partition{
    // at least one CPU in the virtual partition
    int numCPU;
    cpu : CPU[numCPU]; satisfy numCPU >= 1;
    // A virtual partition has its own oS Image
    osImage: OperatingSystemImage;
    // software images may be installed in a virtual partition
    swImage: SoftwareImage;
}
// a nPartition can have virtual partitions
nPartition extends Partition{
    // at least 4 CPUs in a nPartition
    int numCPU;
    int numOfvPars;
    cpu: CPU [numCPU];satisfy numCPU >= 4;
    vpars: VirtualPartition [numOfvPars];
    satisfy numOfvPars > = 0;
    // total number of CPUs in the virtual partitions
    // and the nPartition is the same
    satisfy (numOfvPars > 0) ==>
            (numCPU == (+ i: 0<=i< numOfvPars: vpars[i].numCPU)) ;
}
// A Server can have a number of either nPars or vPars
Server{
    int numCPU ;
    cpu: CPU[numCPU]; satisfy numCPU > 0;
    manufacturer: String;
```

```
    memory: int; // in GB
    model: any;
    int numOfPartitions;
    partitions: any[numOfPartitions];
    parttitionType: any;
    satisfy partitionType <: nPartition || partitionType <: VitualPartition;
    satisfy if partitionType == nPartition then partitions <: nPartition[];
    else partitions <: VitualPartition[];
    satisfy numOfPartitions >= 0;
    // if there are partitions
    satisfy numOfPartitions > 0 == >
        (numCPU == (+ i:0 <= i < numOfPartitions: partitions[i].numCPU) ;
}
HPrpServer extends Server{
    enum hprpTypes {rp8400, rp7410};
    satisfy model \in hprpTypes;
    // only nPartition can be created on this type of server
    satisfy (partitionType <: nPartition);
    // At max of 2 nPartitions may be created
    satisfy (numOfPartitions >= 0 && numOfPartitions <= 2);
    // maximum 16 virtual partitions
    satisfy (model == rp8400) ==>
        ((+ i: 0<= i < numOfPartitions: partitions[i].numOfvPars) <= 16);
    // maximum of 8 virtual partitions
    satisfy (model == rp7410) ==>
     ((+ i:0 <= i< numOfPartitions: partitions[i].numOfvPars) <= 8);
}
Superdome extends Server
{
    enum SuperdomeTypes {16way, 32way, 64way};
    satisfy model \in SuperdomeTypes;
    satisfy (model == 16way) ==> (numCPU==16) && (memory == 64);
    satisfy (model == 32way) ==> (numCPU==32) && (memory == 128);
    satisfy (model == 64way) ==> (numCPU == 64) && (memory == 256);
    satisfy (partitionType <: nPartition) || (partitionType <: VirtualPartition);
    // number of CPUs in a partition is recommended to be 8
    satisfy numOfPartitons > 0 ==>
        ((+ i: 0<= i< numOfPartitions: partitions[i].numCPU) == 8);
    // number of virtual partitions in an nPartition is recommended to be 8
    satisfy (numOfPartitions > 0 && partitionType <: nPartition) ==>
     (forall i :0<= i < nPartition: partitions[i].numOfvPars == 8));
}
main {
    server: Server;
    satisfy server.partitionType <: VirtualPartition;
    satisfy server.numOfPartitions == 4;
}
```

Figure 31: Example for policy for hardware and software partitions in a server.

This example shows how relatively complex constraints on creation of resources can be easily specified as construction policies. Additionally, because the rules specify the maximum number of partitions that can be created on an underlying resource, the policy system will automatically restrict the number of such transient resources.

7.6.5.3 Multi-function and Polymorphic Resources

Multi-function devices and polymorphic devices are resources that need special consideration. Multi-function devices are devices that have multiple capabilities and are capable of performing many functions simultaneously, while polymorphic devices are those that are reconfigurable so that at any given point in time, they can appear as one of a number of different resources. For a given request, multi-function and polymorphic devices may be configured to take on the appearance of one or more resources required for that request.

In the following example a Virtual Service Switch (VSS) is described that is a multi-function device uses Virtual Service Modules (VSM) to offer multiple capabilities. VSM are software modules that can be loaded dynamically an existing VSS.

```
<<code>>
// The software virtual service modules that may be installed on a
//VirtualServiceSwitch are Firewall, Intrusion Detection and Prevention
//(IDP), VPN, Virtual Global Server Load Balancer, Virtual Server Load
//Balancer, Virtual SSL, Web Acceleration
    VirtualServiceModule extends ManagedElement {
    enum types {firewall, idp, vpn, vgslb, vslb, VirtualSSL, webAcceleration};
    vsmType: types;
    vsm: any;
    // In the most generic case, assuming that Firewall, IDP, VPN, VGSLB,
    // VSLB, VirtualSSL, WebAcceleration Resource Types have been already
    // defined separately and they all extend ManagedElement
    satisfy (vsmType == firewall) ==> (vsm <: Firewall);
    satisfy (vsmType == idp) ==> (vsm <: IDP);
    satisfy (vsmType == vpn) ==> (vsm <: VPN);
    satisfy (vsmType == vgslb) ==> (vsm <: VGSLB);
    satisfy (vsmType == vslb) ==> (vsm <: VSLB);
    satisfy (vsmType == VirtualSSL) ==> (vsm <: VirtualSSL);
    satisfy (vsmType == WebAccel) ==> (vsm <: WebAcceleration);
}
// A virtual Service Switch is a polymorphic device and may have any
// combination or all of the modules installed e.g. Inkra Virtual Service Switch

VirtualServiceSwitch {
    number : int ;
    installedVSM: VirtualServiceModule[number];satisfy number > 0;
}
// A Virtual Service Switch may be configured in whichever way, in
// that sense it behaves like a multi-function device. If there was a
// resource type defined that was often used we can pre-define the
// Virtual Service Modules on it
FirewallVPNServiceSwitch extends VirtualServiceSwitch {
    satisfy number >= 2;
    satisfy (installedVSM[0].vsmType == firewall);
    satisfy (installedVSM[1].vsmType == vpn);
}
```

Figure 32: Example for multi-function and polymorphic resources.

7.6.5.4 Class-of-service-based Resource Selection

It is important to allocate resources to users based on certain criteria. These criteria may relate to the class of the user or the corresponding QoS implications. Often simple classes of users are defined (e.g. platinum, gold, silver etc) and users are provided with QoS guarantees based on that classification. The next example demonstrates how different types of servers could be assigned to satisfy a request based on the user classification. In the example, the user class is contained in a context, which is modeled as a policy element that also contains the request. This enables the policy engine to account for the user classification during component selection.

<<code>>

```
// context defines the user context within which the user request is made
Context {
    userType: any;
    enum userTypes {platinum, gold, silver};
    request: any;
    // for silver users, satisfy server requests with basic servers
    satisfy (userType == silver) && (request <: RequestForServer) ==>
        request.grade == basic;
    // for gold users, offer a medium grade server
    satisfy (userType == gold) && (request <: RequestForServer) ==>
        request.grade == medium;
    // for platinum users, provide an advanced grade server
    satisfy (userType == platinum) && (request <: RequestForServer) ==>
        request.grade == advanced;
}
// The following maps different grades of servers to different requests
RequestForServer {
    enum grade {basic, medium, advanced};
    server: any;
    // a basic server is provided for basic grade requests
    satisfy (grade == basic) ==> (server <: Server) && (server.grade == basic);
    // A simple advanced server is provided for medium grade requests
    satisfy (grade == medium) ==> (server <: Server) && (server.grade == advanced);
    // A fail-over advanced grade server is provided for advanced grade requests
    satisfy (grade == advanced) ==>
         (server <: FailOverServer) && (server.grade == advanced);
}
// base definition of a server
Server {
    grade: any;
    model: String;
    processor: any;
    numOfProcessors: int;
    memory: any;
    memtypes: any;
    powerSupply: any;
    powerSupplyTypes: any;
}
// Proliants can offer different capabilities to satisfy different grades
Proliant8500Server extends Server {
    satisfy grade \in {basic, advanced};
```

```
    // scaling up of ProliantServer
    satisfy numOfProcessors \in {1, 4, 8, 16};
    satisfy memtypes  \in
       {onlineSpareMemory, HotPlugMirroredMemory, HotPlugRAIDMemory};
    satisfy powersupplyTypes \in {usual, redundantPowerSupply, UPS};
    // advanced grade Proliants have the following capabilities
    satisfy (grade == advanced) ==>
        (memtypes \in {HotPlugMirroredMemory, HotPlugRAIDMemory}) &&
        (powerSupplyType <: UPS) && numOfProcessors == 16;
    //basic grade Proliants have the following capabilities
    satisfy (grade == basic) ==> (memtypes \in {onlineSpareMemory} ) &&
        ((powerSupplyType <: usual) ||
        (powerSupplyType <: redundantPowerSupply)) && (numOfProcessors <= 8);
}
// a failover server actually contains two servers—one as a backup
FailoverServer extends Server {
    // a failover server
    server: Server;
    backupServer: Server;
}
// goal specified as a request
main{
    context: Context;
    satisfy context.userType == platinum;
    satisfy context.request <: RequestForServer;
}
```

Figure 33: Example for class-of-service based resource selection.

Similar examples can be constructed for other capabilities such as availability, response time, throughput, processor speed based selection of components. These metrics have to be considered when constructing complex resources for different classes of users. Additionally, other metrics that have to be managed at run-time may be specified using a similar constraint language.

7.6.6 Implementation Issues

A prototype resource manager has been implemented to validate the automatic construction of resource topologies discussed in this section. The resource model extends the Common Information Model [CIM], which is an object-oriented information model standard for IT systems from Distributed Management Task Force [DMTF]. Because CIM defines information models for a large number of IT resources (including models for devices, networks, databases, and users), all conforming to a single meta-model, it allows us to rapidly incorporate a large number of resources in the prototype without having to construct resource models from scratch. All resource type definitions map to classes in CIM (typically those under CIM_System class). Many types needed already exist as CIM classes, but others (e.g., appserver, webserver, tiers, e-commercesites etc.) have been added on top of the existing CIM classes.

Because construction policy is associated with the resource type definitions, it is convenient to combine the policy specification for resource construction with the type definition of the resource. In order to validate the approach, policy constraints need to be defined for the existing CIM resource types in addition to the types that were added to the

CIM class hierarchy. However, the CIM meta-model does not provide for associating policy instances with the type definitions. In this work, construction policy constraints are explored that can be specified within the framework provided by CIM.

One possible approach is to represent constraints as properties within CIM classes, but "tag" these special properties as constraints. CIM allows qualifiers for adding such special tags to properties, and a new qualifier called "constraint" can be added on a property to indicate that its value is a constraint as opposed to a typical property. The following example demonstrates how a constraint such as "If the total visible memory size of a machine is less than 10 MB, then its OS cannot be OS/390" using this approach.

```
<<code>>
Qualifier Constraint : boolean = false,
    Scope(property),
    Flavor(DisableOverride);
Qualifier ConstraintLanguage : string = NULL,
    Scope(property),
    Flavor(DisableOverride);
class CIM_OperatingSystem : EnabledLogicalElement {
    [Static, Constraint, ConstraintLanguage("Text")]
    string constraint1 =
        "if TotalVisibleMemorySize < 10000 then OSType != 60";
}
```

Figure 34: Example of Managed Object Format (MOF).

This example is written in Managed Object Format (MOF) – a language used to describe CIM models. The first two Qualifier declarations declare two qualifiers – "Constraint" and "ConstraintLanguage". This has to be done once so that the MOF parser understands these qualifiers. The declaration of the class "CIM_OperatingSystem" includes a property called constraint1, whose value contains the constraint expression. The "Static" qualifier indicates that this constraint holds true for all instances of the class. The "Constraint" qualifier indicates that this property has to be interpreted as a constraint. For simplicity, the constraint itself is expressed in plain text (as indicated through the "ConstraintLanguage" qualifier), although more formal constraint languages such as the one described in this section or the Object Constraint Language [OCL] can be used to represent the constraint expression.

The advantage of this approach is that it does not require a change in the CIM meta-model. All existing CIM and MOF tools (such as parsers, repositories, and browsers) will continue to work with this extension without any modification. However, they will not be able to interpret the meaning of a constraint nor would they be able to parse or check their syntactic correctness. For example, the MOF parser, which parses a MOF file and stores the defined classes into a CIM repository, would treat these constraints as strings and will not check for the correctness of the constraint expression. For a large number of constraints, this approach would make creating and maintaining policies difficult.

Another approach to incorporate constraints into CIM is to change CIM's meta-model. In addition to classes containing properties and methods (as is currently the case), constraints would need to be added as first class entities. This would require changes to the MOF syntax as the following example demonstrates:

```
<<code>>
class CIM_OperatingSystem : EnabledLogicalElement {
   [ConstraintLanguage("Text")]
   satisfy  "if TotalVisibleMemorySize < 10000 then OSType != 60";
}
```

Figure 35: Example of a constraint expression added to a class.

The example shows a constraint expression added to a class in a more natural manner using a "satisfy" keyword. Qualifiers can be used to add additional information about the constraints, just like they are used to add additional information to properties and methods. This approach would require changes to existing CIM tools, but is nevertheless a more elegant and complete solution.

In addition, it needs to be explored how the constraints themselves will need to be represented. One possibility is to use OCL. OCL is being proposed to DMTF as an extension to CIM for defining constraints. OCL offers most of the constructs necessary for construction policy; however, like CIM, it is complex. For the prototype, a simpler language is used (with the grammar shown in [Gra05]). However, OCL constraints could have been used in the prototype as well.

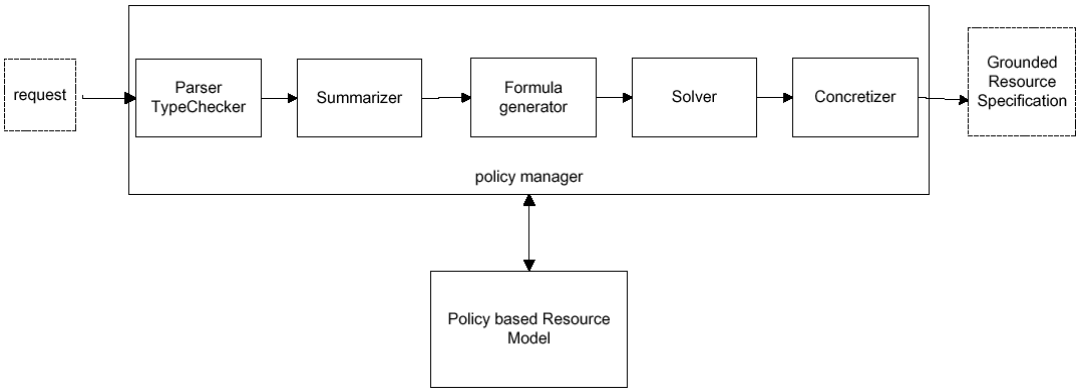


Figure 36: Policy engine as part of the automation tool chain.

A CIMOM based resource inventory has been implemented based on the SNIA CIMOM [SNIA-OM]. In the inventory, the CIM resource model was extended for defining new resource types that are required, have added constraints to the existing and new resource types and have populated the CIMOM with these resource type definitions. A policy engine was used that was developed in HP Labs to solve the constraint satisfaction problem such that, given a set of constraints written in first-order logic with linear arithmetic, it can be determined if the constraints are feasible, and if so, produce a resource description that satisfies the request.

Figure 36 shows the basic architecture of the policy engine. When the request arrives at the policy engine, it is parsed and checked against the resource model types available from the type database. The summarizer captures all relevant resource types that may be necessary to fulfill the request, collects all policy constraints from the different types, as well as from the request, and converts them into an intermediate representation. The formula generator uses these constraints to compose logical formulae that represent the constraints. The solver reasons on the formulae and performs a feasibility check. If the

solver determines that a solution is feasible it uses the concretizer to create a system specification that satisfies all the constraints [Ram06].

7.6.7 Related Work

Introducing constraints in UML specification of systems for configuration purposes is discussed in [Fel99]. They define a set of construction rules at one place termed a domain. In that sense the approach is similar to expert systems. In the approach, constraints were hierarchically embedded distributing constraints on to various resource types, and taking into account these constraints as the construction happens as opposed to creating a large number of constraints (rules) a priori. This approach enables flexibility and extensibility in specification of constraint and in automatic construction depending on the user requirements. The differing user requirements may result in one construction being different from another.

The ClassAds MatchMaking work [Ram98] assumes that the match-maker matches the requestor entity's request against the provider entity's ClassAds (which are specifications in a semi-structured language). The assumption is that all the resources (like machines) exist a-priori and have been advertised. Some of the resource instances may not exist a-priori (as is the case with transient/virtual resources) or may be logically constructed resources that have to be instantiated on-demand (e.g. appserver/tier/farm/e-commerce site). This causes a problem for approaches that undertake match-making only on instances. The approach here enabled the construction on-the-fly by embedding constraints hierarchically in the resource types as described here. The same concepts are extensible to resource instances as well. It is also not clear whether the ClassAds language supports first-order logic and linear arithmetic. As shown in the examples, it is important to have notions of quantifiers, implications, equivalences and other first order-logic expressions for reasoning.

There is significant work that has been done in the community in terms of specifying, and associating events, conditions and actions for policies, namely IETF [IETFPol], CIM [CIMPol], PARLAY [PARPol], [Slo93], PONDER [Slo01] etc. Additional work relates to using policy for SLA management [Ver01]. These bodies of work have not considered incorporating first-order logic and linear arithmetic based constraints in resource types for automatic constructions of resources and have not used a constraint satisfaction approach for arriving at a constructed resource specification. The WS-Policy [WS-Pol] work at OASIS has focused on generic schemas for specifying arbitrary policy assertions on web services. The constraints as specified in this article may be embedded inside these assertions.

A CIM-based representation has been used to specify resource-level construction policies. We have proposed an intermediate language based on first-order logic with linear arithmetic extensions to enable reasoning and analysis on the policies and goals. As shown in the examples, the language proposed by us is expressive enough to capture a wide variety of resource models and policy behaviors. Once these policies have been specified, resource construction can be framed as a goal-satisfaction problem using the resource models for undertaking resource composition and component selection.

7.7 Summary

Modern data center environments are dynamic environments and deal with a large number of complex heterogeneous resources. These environments have to configure complex resources from other resource components while keeping in mind the user requirements specified as goals, resource requirements/constraints and operator policies.

In this section the research for the Planning and Design Layer of the architecture of a DCI-OS has been examined.

Specifically, these areas have been explored in which research has been conducted:

- Performance Engineering Processes for Data Centers [Rol08],
- Systematic Approach to Derive IT Configurations from Business Processes [Gra08], and
- Resource Topology Design [Gra05].

It was discussed how automatic construction of complex resource environments can be achieved in a more systematic manner by using experimental and automated methods, for instance, by embedding constraints in the resource models themselves and using a constraint satisfaction approach to solve the relevant constraints coming from multiple sources. More and more advanced techniques are needed to allow better support for decisions made in the data center planning stages.

Chapter 8

The Infrastructure Services Layer

The Infrastructure Services layer primarily addresses the side of infrastructure services with their main task to provide the execution environments for Application Services.

Figure 37 shows the layer as it was presented in the Architecture in Chapter 5. An instance of an *Infrastructure Service* is shown as the set of configured resources within the blue rectangular shape at the top of Figure 37. These resources have been acquired from the data center resource environment and configured according to the specifications provided by the *Resource Topology* which is a blueprint (a model) of the Infrastructure Service defining the types and quantities of resources needed along with the proper configurations as needed by the Infrastructure Service to support the Application Service.

The Resource Topology is defined as a model using the Common Information Model (CIM) [CIM].

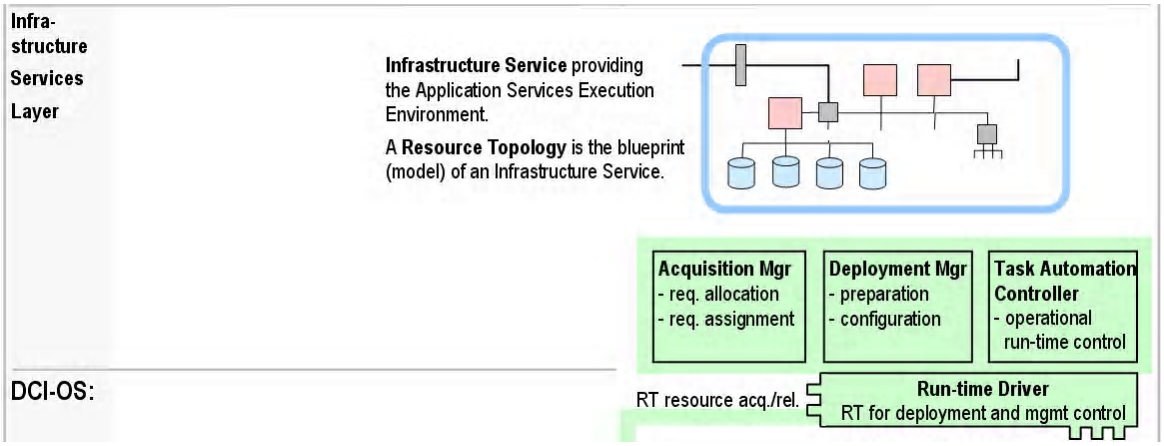


Figure 37: Infrastructure Services Layer with run-time support system.

As part of the DCI-OS, the Infrastructure Services Layer provides support systems which allow acquiring and releasing resources from the data center-part of the DCI-OS as well as deploying configurations onto those resources fitting the needs of the Infrastructure Service. Resource acquisition and release as well as deployment tasks must be coordinated in order to occur in the proper sequence.

These tasks are performed by three main modules,

- The resource *Acquisition Manager* – allowing the Infrastructure Service to request and release resources of specified types and quantities;
- The *Deployment Manager* – allowing specific configurations to be deployed onto freshly acquired resources; and
- The *Task Automation Controller* – coordinating the various activities which have to occur when new resources are provisioned as well as coordinating the overall lifecycle control tasks of the Infrastructure Service (create, start, suspend, resume, stop, destroy).

More modules can be added as needed. The *Run-time Driver* module shown in Figure 37 establishes the connection with the Data Center-related part of the DCI-OS using web services interfaces and protocols.

This architecture allows an Infrastructure Service to be replicated within one data center (multiple instances are created) as well as migration of the Infrastructure Service into other data centers. Purpose of the Infrastructure Services Layer is to provide modules which are widely self-contained such that they can be recreated easy.

The purpose of the Infrastructure Services layer is similar to the run-time system in operating systems. While an operating system run-time library also allows acquiring and releasing resources from the underlying operating system, the task spectrum in the Infrastructure Services layer is significantly broader due to the broader spectrum of resources. In an operating system, considered resources are simple resources such as memory, processor shares or external storage. For Infrastructure Services, resources encompass entire machines, storage devices, and networks requiring additional application of service-specific configurations which is performed by the Deployment Manager in the Infrastructure Services layer.

Unlike in an operating system where the run-time system is part of the same process it controls, the modules of the Infrastructure Services layer execute on ***separate management machines*** which are dedicated to the DCI-OS. Reason for this decision is to avoid interference between resources (e.g. machines) where Application Services execute and where management functions execute. It is particularly important that the resources of the management system remain available and functional when the resources of the Infrastructure Service have been suspended or removed.

This chapter will discuss three particular modules of the Infrastructure Services layer:

- the *Task Automation Controller*,
- the *Deployment Manager*, and
- the *Resource Acquisition Manager*.

The Task Automation Controller is discussed because it demonstrates a principle of declarative control over coordination tasks, which has been found beneficial over traditional workflow-based coordination implementation (e.g. with workflow engines or script executions). This declarative approach follows the pattern of a *Controller* which allows setting a state (the declaration of *what* should be) as a Desired State and the controller providing the logic *how* this state is achieved and maintained.

This is an advanced approach to coordination in management which often relies on describing control flows (workflows, processes, scripts, etc.), which execute well as long as no problem occurs. If a problem occurs, it is hard to identify where the flow broke, which state has been altered and how to reset the affected components. Using the declarative approach of a controller over specifying control flows has been extremely beneficial and is one of the key insights of the development of the DCI-OS

The case of the Deployment Manager is discussed for another reason. It is presented to demonstrate the integration of an existing management system into the architecture of a DCI-OS for a specific role and a specific purpose. It cannot be assumed that all functionality a DCI-OS requires will be redeveloped from ground up. Rather, a DCI-OS will act as an integration platform of the various tools and systems which are in existence today bringing their capabilities together for achieving deep, integrated automation capabilities which cannot be achieved by individual management systems operated in isolation. In this chapter, the case of integrating an existing deployment system (called Radia) into the architecture of the DCI-OS is presented.

The task of the *Resource Acquisition Manager* is to provide the access to the data center resources by communicating with the DCI-OS Resource Management Layer. The Resource Acquisition Manager also follows a controller pattern where a certain level of resource supply is constantly compared with the actual resource usage consumed by the Infrastructure Service. It is an example of managing resources in a dynamic manner (like it is the case in an operating system) in a data center rather than statically where resources such as servers are acquired and dedicated to a specific application which cannot be changed during use. Managing resources more dynamically and more effectively in a data center has been a major requirement for a DCI-OS.

The discussion begins with the Task Automation Controller in section 8.1, continues with the Deployment Manager in section 8.2 and finished with the Resource Acquisition Manager in section 8.3.

8.1 The Task Automation Controller

This section presents the design and implementation of a specific module of the architecture of a DCI-OS. This module, the Task Automation Controller, implements task automation using the pattern of a controller which operates upon discrete management states of an infrastructure service. The Task Automation Controller allows setting policy in form of a so-called desired state in which an infrastructure service should be maintained. Deviations from the desired state are detected by comparing the desired state with a so-called observed state, and counter-actions are initiated automatically. If automatic correction is not possible, such as in case of physical failure, events are generated and propagated up in the management hierarchy.

The Task Automation Controller has been presented at IM'2007 [Gra07]. A novel approach of this controller is that it uses an implementation based on Petri Nets in combination with allowing setting control policy in terms of discrete management states such as "ready" or "operational". An engine instantly starts executing the Petri Net when state in either the desired or the observed state sets changed. Prior work to this controller was presented in [Gra04], [Gra05a,b] and [Col05].

The section first gives a general introduction to IT task automation and then presents how controllers are used to automate IT management tasks.

8.1.1 Automation in IT Management

Controllers in computer systems have mainly been explored for automating regulative tasks such as admission control or resource supply control. The majority of IT management tasks, however, rely on discrete management states and coordinated transitions between those states.

The section shows how the concept of a feedback system can also be applied to automate operational management tasks. The section introduces the concept and a realization of a Task Automation Controller, which operates on discrete management states expressed as a pair of models for desired and observed state. Models are represented as a special form of Place-Transition Nets (PTN or Petri Nets). Controller logic directly executes PTN in order to achieve and maintain alignment between desired and observed state in a managed domain. In contrast to workflow systems, PTN combine the description of state and actions in one model (graph).

Three operational database management tasks have been implemented as a proof of concept in a blade server automation infrastructure using the Task Automation Controller.

The reality of IT management is dominated by a legacy of management systems which have been designed as tools for human operators, not for automation and self-management. Today, IT management can be seen as at a stage of mechanization where operators use management systems as tools to carry out management tasks. Management systems are designed as tools for operators and facilitate operations through consoles. Some tools, and lately more and more tools, also provide API to allow programmatic control and scripting for integration in process automation chains.

One can identify three stages of automation which are found in IT management today.

- *The 1st Stage of Automation: Scripts and Workflows.* Automation at this stage is characterized by management tools that can be accessed through API or command lines enabling scripts and workflows to describe action sequences of repeatable management tasks. The operator initiates the script or workflow execution as opposed to actions individually.
- *The 2nd Stage of Automation: Policies.* Initiation of action sequences can be triggered by conditions reported as events from the managed system. Definitions of Event-Condition-Action (ECA) triples are also often referred to as ECA polices. As events are reported from the managed environment, they pass through a sequence of conditions, and for each condition evaluating to true, the associated action sequence is executed. ECA policies are widely used IT management [Slo01].

First and second stage automation have no knowledge about the changes executions cause in the managed environment. An action sequence runs once when initiated. There is no inherent ability to detect whether the goal which caused the execution actually has been achieved in the managed environment or not.

To some extent, conditions in ECA policies can be seen as representations of a desired state such as thresholds that should not be passed. However, an ECA system relies on

external events to trigger evaluation for executing actions, which is the difference to a controller that autonomously evaluates conditions and triggers actions in order to maintain a managed environment aligned to its desired state.

- *The 3rd Stage of Automation: Controllers.* A controller has a description (model) about a desired status of its controlled domain. It also has a reflection (model) of the current status that is observed from the controlled domain. Both models, which are called the *Desired State Model (DSM)* and the *Observed State Model (OSM)*, are constantly evaluated by the controller. Corrective actions are deducted and executed as differences occur between the two models.

Intended change in the controlled domain is achieved by changing the desired state model, either manually by an operator or programmatically by another system or controller. Unintended change can occur any time the system that is reflected back into the observed state model such as in case of a failure. Both kinds of changes may initiate actions in the controller in order to maintain alignment between observed and desired state. Controllers may not be able to achieve alignment under all conditions. Those cases need to be detected and reported to a superior instance as uncorrectable conditions

8.1.2 The Controller Concept in IT Management

In general, a controller adjusts conditions of a controlled (or managed) element or domain by altering control knobs as a function of measured parameters and controller settings. Measured parameters can be interpreted as a form of observed state. Controller settings can be seen as a form of desired state. Controller settings or desired state represents a goal according to which the controller aligns the controlled element by adjusting its control knobs according to the result of the evaluation of the control function. A large body of literature exists on feedback control in computing systems [Hell04].

Controllers have been introduced to computer systems in a variety of ways:

- *Regulative Controllers* operate based on numeric input for measurements and settings to their control functions, which produces numeric output for control knobs in the controlled environment. Examples are admission controllers, which throttle incoming workload when it surpasses processing capacity [Hell01], or flex controllers, which expand or shrink resource supply based on workload [Wang05]. Controller settings can only be altered from outside, not by the controller itself.
- *Adaptive Controllers* can alter (tune) the control settings or even the control function as result of reasoning upon observed behavior in the past and deriving predictions for the future [Ast94], [Xu06]. Adaptive admission controllers have been shown in [Wels03]. Adaptive flex controllers have been presented in [Liu05].
- *Autonomic Manager* is a basic concept of Autonomic Computing [Kep03]. It defines a closed loop with stages: monitor, analyze, plan, execute for a managed element or domain. A number of controllers have been implemented based on this concept, mainly regulative and adaptive controllers [Smith04].
- *Task Automation Controller* which is discussed in this section is similar to a regulative controller. In contrast, it is not based on numeric control parameters,

settings and function; it uses two discrete state models associated with a managed domain, the desired state model and the observed state model. The control function represents discrete state logic that evaluates the two models and produces a sequence of actions based on differences. A Task Automation Controller also meets the general criteria of the Autonomic Manager concept. It is a specific form directed to IT task automation and composition of automated IT management process chains.

8.1.2.1 Problems With Workflow Systems

The modeling framework determines the expressiveness of models. It also determines the mechanisms that are required for interpretation (the controller logic in this case).

Using sole declarative models entirely hides the logic for interpretation inside the controller. A declarative model only describes a desired or observed state (data). It does neither describe how this state should be interpreted, nor how it came to this state and what should happen in that state. Current model-driven approaches to IT management favor the use of declarative models [Thom05].

While this seems desirable at a fist glance, it has a number of shortcomings:

- Logic is built into controllers, typically hard-coded and cannot be customized.
- Logic is not modeled, hence remains unclear and hard to trace.
- Since logic depends on the structure and semantics of models, changes will likely break the controller logic requiring code replacement in controller instances.
- Controllers depend on interactions with the managed environment. Sole declarative models do not provide means to describe those interactions and dependencies.
- Controller composition and automated IT management process chains require coordination among controllers. Again, when logic is built into the code of the controller, it cannot be customized making automated IT management process chains difficult to build and maintain.

Using declarative models in combination with built-in controller logic may be desirable for lower-level resource controllers with a fixed behavior. It is not sufficient for higher-ordered controllers that operate at a level of automated IT management process chains that require customization and adaptation in a customer environment.

8.1.2.2 Alternatives

To overcome the problem of separation of the model from its interpretation (logic), models can be supplemented with the interpretation logic to avoid hard-coded logic in controllers. Some modeling frameworks support the representation of interpretation rules. An example is the Resource Description Framework (RDF). A rules engine like the Jena Rules engine could execute the model interpretation rules that are part of the model. However, dynamic behavior is hard to integrate in rule engines.

Workflows are typically used to describe configurable, dynamic behavior across systems and execute on it. However, it is difficult to represent state in workflows, such as observed or desired state of a managed element. Once again, it leads to the separation

between "state models" and "execution logic", although the execution logic is now configurable in a workflow engine and not built into the controller.

Another aspect with workflow languages such as BPEL is that they are designed for business transactions, which may suit higher-level, more transactional IT management processes, but is not a good match for the asynchronous and partially unpredictable behavior that occurs in a dynamic management environment such as asynchronous events, race conditions or critical sections.

A balance needs to be found that brings all those aspects together: the representation of desired and observed management states, the description of dynamic interactions and dependencies with other controllers, and the representation of interpretation logic in a form which allows to execute on a generic engine as opposed to built-in code in controllers.

8.1.3 Task Automation Controller Design and Implementation

For the Task Automation Controller, initial ideas of a Service Delivery Controller [Col05] have been taken and generalized. The general components of the Task Automation Controller are shown in Figure 38 with the following components:

- DSM Interface through which the desired state model is accessed (read/write);
- OSM Interface through which the observed state model is accessed (read only, subscriptions to change events);
- Controller function (logic) which consists of:
  - Differencer logic which compares the desired and the observed state model;
  - Action sequencer logic which derives actions from the difference;
- Observer Connector through which the observed state model is updated from the managed environment (polled by the controller or event-based);
- Actuator Connector through which actions are passed into the managed environment for execution.

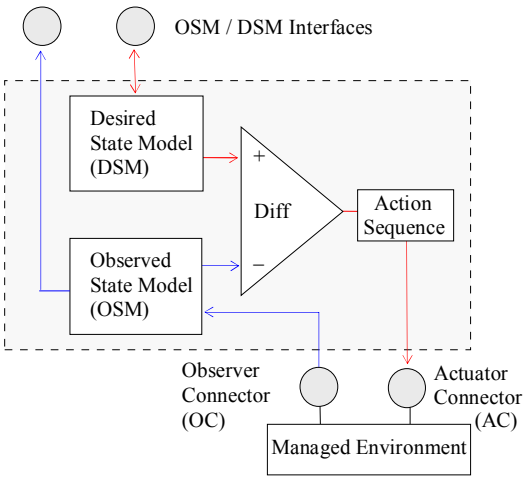


Figure 38: Overview of the Task Automation Controller.

Control information flows into the controller in form of desired state definitions and changes to those definitions through the DSM interface. It also flows into the controller

as changes to the observed state model through the observed connector. The controller's internal control loop aims to maintain the alignment between observed and desired state by deriving and issuing actions through its actuator connector. If this cannot be achieved, superior instances can subscribe to event types at the observed state model interface to be notified when those conditions occur.

8.1.3.1 Approach: Petri Nets

Place-Transition Nets provide a good approximation:

- State can be represented in a Place-Transition Net,
- Interactions with managed environment causing state changes can be expressed,
- Dynamic behavior and coordination with other controllers can be represented,
- Synchronous and asynchronous interactions can be modeled,
- Place-Transition Nets can be executed (interpreted) by a generic execution engine.

Place-Transition Nets are well proven in domains such as manufacturing and supply chains. They have also been used in telecommunications for modeling and verifying protocols. A large body of experience and knowledge exists, from formal techniques for proving liveliness or reachability to simulation environments. However, despite favorable properties, PTN have not widely been leveraged in IT management and automation, which may partially be due to a lack of tooling and experience with PTN in the domain of IT management.

The realization of the Task Automation Controller presented in this section applies Place-Transition Nets as the modeling framework for representing the models for desired and observed states and for executing controller logic by interpreting PTN. A generic Place-Transition Net execution engine was built and included in controllers replacing their hard-coded logic. All interactions with the managed environment as well as with other controllers are driven by interpreting PTN.

8.1.3.2 Place-Transition Nets (PTN)

Place-Transition Nets (PTN) or Petri Nets were first introduced in [Petri62]. A Petri Net is defined as a 6-tuple (S, T, F, M_0, W, K) where S is a set of places and T is a set of transitions. F is a set of arcs between either a place and a transition or a transition and a place: $F \subseteq (S \times T) \cup (T \times S)$. A token is a construct that represents state in a place.

A distribution of tokens over the places in a net is called a marking. M_0 is the initial marking, $M_0 : S \rightarrow \mathrm{N}$ with each place $s \in S$ having $n \in N$ initial tokens. $W : F \rightarrow N$ is a set of arc weights $n \in N$ assigned to each arc $f \in F$ denoting how many tokens are consumed from a place by a transition and how many tokens are produced by a transition and added to a subsequent place. $K : S \rightarrow N$ is a set of capacity restrictions which assigns to each place $s \in S$ some positive number $n \in N$ denoting the maximum number of tokens that can occupy that place. A net in which each of its places has some capacity k is known as a k-bounded Petri Net.

Places may contain any number of tokens up to the capacity restriction $k \in K$. A marking is altered $m_i \rightarrow t_{i,j} \rightarrow m_j$, $m_i \in M$, $m_j \in M$ when transition $t_{i,j} \in T$ fires. Firing a transition is an atomic operation.

Transitions *may* fire, when they are enabled. Transitions are enabled when they have at least the amount of tokens in each input place specified by the inbound weight of the transition (default is 1). When a transition fires, it consumes the weight amount of tokens from each inbound place and adds the amount of tokens specified by the outbound weight to each outbound place (default is 1).

These fundamental properties of Petri Nets allow reasoning on properties such as reachability, liveness or boundedness.

Execution of Petri Nets is nondeterministic. Multiple transitions can be enabled at the same time, any one of which can fire in any order or simultaneously. Transitions may not fire immediately when they become enabled or may not fire at all. Since firing is non-deterministic, Petri Nets are suited for modeling asynchronous and concurrent behavior of distributed systems [Pett81].

However, some assumptions must be made in regard to non-determinism for the practical use of PTN for task automation. Furthermore, a combination of three extensions Colored Petri Nets (CP-Nets), Hierarchical Petri Nets and Timed Petri Nets is used.

8.1.3.3 Colored Petri Nets (CP-Nets)

In a basic Petri Net, tokens are indistinguishable ("black") and themselves stateless. Only their assignment to a place at a time determines the state (marking) in the network.

A number of examples in the domains of network protocols and manufacturing supply chains are shown in [Jen98] where distinguishable items travel through a network as tokens following the PTN rules. Those items (represented as tokens) must carry own state ("color") in order to be distinguishable. Prof. Kurt Jensen from the University of Aarhus has developed Colored Petri Nets [Jen97] by introducing following extensions:

- assign state (a value) to tokens that is defined by a simple or complex type,
- assign a type to places determining the type of tokens it can hold,
- allow multi-sets of tokens of same type and value by specifying coefficients, and
- assign expressions (functions) to arcs that can be bound to token values and evaluated when tokens pass through transitions during firing.

Transitions in a CP-Net thus do not only alter the marking of the overall net and bring tokens to other places. Evaluation of arc expressions also allows altering the state within tokens when they pass through a transition. Those functions can alter token state.

Tokens do not share their states. States of multiple tokens can be combined as result of evaluating arc expressions when they are part of the same transaction and hence part of the same evaluation process. Altering states in tokens by evaluating arc expressions allows "programming" in a CP-Net. Tools have been developed for CP-Nets that are widely used, such as CPNTools [CPN].

8.1.3.4 Hierarchical Petri Nets

The idea behind hierarchical Petri Nets is to introduce scope, reusable building blocks and a modular structure in larger nets. Each place can be expanded into a (sub-) net into which tokens flow via inbound transitions, internally travel through the subnet and finally return or produce tokens in the surrounding net. Nets and places within nets can be made self-similar such that they can be composed hierarchically [Hub91].

Inbound and outbound arcs to a (subnet-) place also define the interface to an underlying net. The structure of this net can remain hidden as long as the interface is known. Properties of Hierarchical Petri Nets and examples of reusable subnets (such as for critical sections or the reader-writer problem) are discussed in [Cho82].

8.1.3.5 Timed Petri Nets

Petri Nets are non-deterministic in terms of when enabled transitions fire or if they fire at all. Again for practical reasons, Timed Petri Nets allow to define an interval within which an enabled transition must fire. The lower bound of the interval defines the minimal and the upper bound the maximal time an enabled transition must or can wait to fire.

8.1.3.6 Combination of Colored, Hierarchical, Timed Petri Nets

A combination of the three Petri Net extensions has been chosen as foundation for the PTN used in the Task Automation Controller. In addition, following assumptions are made, which are explained later in the text:

- Two types of places are introduced: regular places and connector places.
- Two types of tokens are introduced: regular and activity tokens.
- Two special transition rules are introduced called bonding and detaching.
- The ambiguity of a conflict ("confusion") in a PTN is resolved by labeling outbound arcs from places with disjunctive values and introducing a choice field as part of a tokens data type. In case of a conflict, the outbound arc with a matching choice label determines the next enabled transition.
- CP-Net multi-sets are not allowed.
- The default firing interval for transitions is [min=0, max=0], which means that transitions fire immediately as soon as they become enabled. The firing order of multiple simultaneously enabled transitions is arbitrary (undefined). The firing interval can be redefined for transitions.

8.1.3.7 Workflow Patterns Expressed in Petri Nets

Figure 39 shows common workflow patterns in terms of PTN. Case (a) shows a simple sequence. Since the default weight of arcs is 1, the token in place s_1 enables transition $t_{1,2}$. Firing $t_{1,2}$ brings the token to place s_2 by reducing the number of tokens in s_1 by 1 and increasing it by 1 in s_2. Case (b) is similar, except that 1 token is added to both places s_2 and s_3. One token from s_1 becomes duplicated in places s_2 and s_3. Both tokens in s_2 and s_3 are independent, which semantically corresponds to forking a process.

In case (c), transition $t_{1,2,3}$ is only enabled when both inbound places have at least one token each. Following the normal transition rule, 1 token is removed from each inbound place s_1 and s_2 and 1 token is added to s_3. This means, two tokens from s_2 and s_3 join at this transition. Two independently traveling tokens are synchronized.

The literature refers to case (d) as conflict or as "confusion" because both transitions $t_{1,2}$ and $t_{1,3}$ are enabled. Only one transition can fire since the one enabling token cannot be reduced twice by two firing transitions. Classic Petri Nets define this case as non-deterministic choice for selecting the firing transaction. One common approach to turn this case into a deterministic choice is to label outbound arcs (such as with "success" or

"failure" in the figure) and determine the firing transition by computing a result against which the labels are compared. This results into the known branching pattern.

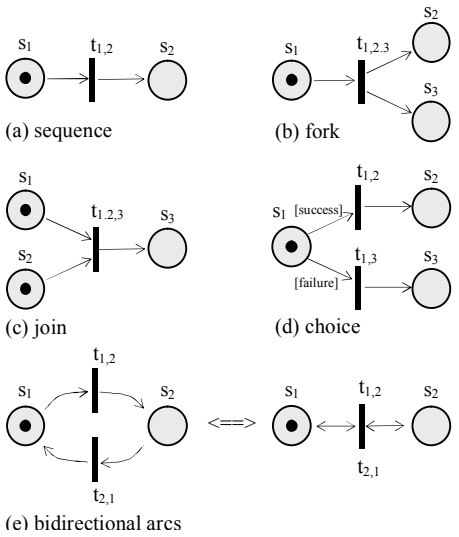


Figure 39: Basic workflow patterns as Petri Nets.

Case (e) shows a convention which is often used in Petri Nets to abbreviate bidirectional arcs. Both notations are semantically equivalent. Note that the bidirectional transition actually represents two transitions.

8.1.4 Representing Desired and Observed State Models as PTN

Figure 40 shows a simple lifecycle model for class of servers as pair of PTN for desired and observed states. A pair of tokens (*md, mo*) represents the managed element, which is a particular server instance. Position of token *md* in a place in the desired state model represents the desired status of the managed element. Position of token *mo* in a place in the observed state model represents the observed status of the element in the managed environment.

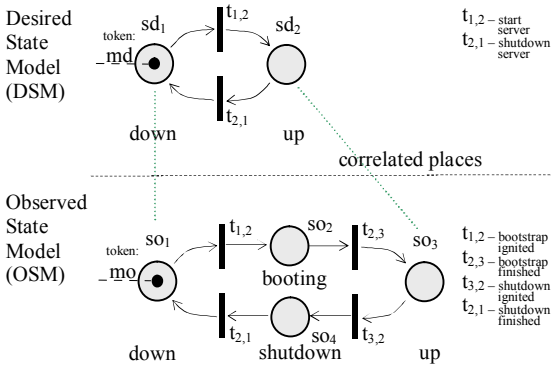


Figure 40: Simple lifecycle models for a server expressed as Petri Nets.

(The transitions in Figure 40 are not yet controlled.)

A token *md* in place sd_1 in the desired state model (as shown in the figure) means that the server is supposed to be up. The token in sd_2 would mean that the server is desired to be down. A token *mo* in place so_1 in the observed state model indicates that the server is

observed as down (as shown in the figure). A token in so_2 would mean the server would be in the process of booting. The token in so_3 would mean that the server would be observed as up, and the token in so_4 would indicate the server is shutting down.

There are correlations between certain places in the desired and the observed state models such as sd_1 and so_1 [down] and sd_2 and so_3 [up]. Those correlated places represent alignment between desired and observed state when the two tokens (*md, mo*) reside in those correlated places. Correlated places represent the same management status, either as desired or observed. Correlated places are defined as one-to-one relationships between pairs of places from the desired state model: $S_{DS} = \{ sd_i \}$, $i=1...n$ and from the observed state model $S_{OS} = \{ so_j \}$, $j=1...m, j \geq i$.

A set of correlated places C is defined as set $C = \{ (sd_i , so_j) \}$ with $\forall\, sd_i \in S_{DS} : sd_i \rightarrow so_j$ and $so_j \in S_{OS} : sd_j \rightarrow so_i$. For each place in a desired state model, there must be a correlated place in the observed state model. There may be more places in the observed state model that represent intermediate stages of a managed element (such as booting). The correlated places for the example in Figure 40 are: $C = \{ (sd_1 , so_1), (sd_2 , so_3) \}$.

A managed element is *aligned* when its pair of tokens (*md, mo*) is residing in a pair of correlated places. The subset of markings that represent alignment between desired and observed states is $M_{ALIGN} : md \rightarrow sd_i,\ mo \rightarrow so_j$ with $(sd_i, so_j) \in C$. Any other marking represents non-alignment between desired and observed state for the managed element.

8.1.5 Petri Net Interpretation of the Controller Logic

In context of the Task Automation Controller, two main domains are modeled as PTN: one is the model of Desired State (DS) and one is the model of Observed State (OS) for a managed environment.

A *place* represents a desired or observed state in the managed environment. Examples of such states are: [system is down], [server is down], [application is running], or [maintenance is in progress]. States of a typical lifecycle diagrams correspond to places in a PTN. (A notation is used in the following for describing [states] and <transitions>).

A *transition* represents a change between those states. When the prior state was [server is down], and the subsequent state in the model is [server is up], then the transitions between the two states is <boot>. Since booting of a server is a longer term operation, it by itself can be modeled as a state: [server is booting]. It is good practice to model rather "short" or "timeless" indications as transitions such as <ignite server boot>, followed by the [server is booting] state (place), followed by a transition <bootstrap finished successfully> before entering the [server is up] state.

Figure 41 shows the expansion of a regular PTN transition into a transition state s_2, which is a regular PTN place. The transition $[s_{1=DOWN}] \rightarrow <t_{1,2=BOOT}> \rightarrow [s_{2=UP}]$ expands to $[s_{1=DOWN}] \rightarrow <t_{1,2=IGNITE\ BOOT}> \rightarrow [s_{2=BOOTING}] \rightarrow <t_{2,3=BOOT\ FINISHED}> \rightarrow [s_{3=UP}]$.

A *token* represents a managed element. The place in which a token resides defines the (either desired or observed) status of the associated managed element. Since tokens can carry own state, multiple managed entities (e.g. multiple servers) can be represented and transition in the same PTN, each independent from the others. Each managed element is represented by a pair of tokens, one representing its desired and one representing its observed state in the two PTN models, respectively.

The equivalent of a token in the workflow language BPEL would be a BPEL message. However, messages in workflow languages are meant to be received and processed according to the workflow definition.

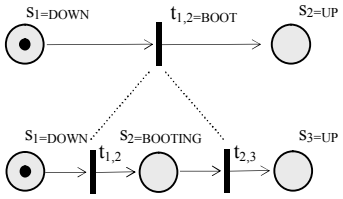


Figure 41: Expansion of a transition into a transition state.

8.1.6 Deriving Actions from Desired State Changes

A state of alignment can only change when either new desired state is defined or when a change occurs in the managed environment that reflects back to a change in observed state. Both changes lead to non-alignment since at least one token must transition from the correlated pair of places to another place, and one place can only be in one correlation according to the definition of *C*.

In case desired state is changed (intended change), the controller must determine a sequence of actions that brings the managed environment into new alignment. In the example, when desired state of a server is changed from [down] to [up], the boot process must be ignited and completed before the observed state can indicate that the server is up.

Aligned state: $c_{1=DOWN} = (sd_1, so_1)$.

1.) $md \rightarrow sd_2$ with $t_{1,2}$ firing in DSM leading to:

2.) $mo \rightarrow so_2$ with $t_{1,2}$ firing in OSM (igniting the server boot and booting the server),

3.) $mo \rightarrow so_3$ with $t_{2,3}$ firing in OSM (server boot completed).

Aligned state: $c_{2=UP} = (sd_2, so_3)$.

Server shutdown follows the same pattern. It will later be shown how error conditions can be taken into account and eventual corrective actions can be derived from error states. Errors and failures are examples of unintended changes that may occur in the observed state model.

All activity of the controller depends on firing transitions in DSM and/or OSM. In order to control firing, DSM and OSM are extended by connector places and activity tokens.

8.1.6.1 Connector Places

Connector places supplement DSM and OSM to provide the interfaces with the environment. Tokens can be generated or consumed in connector places as effects of interactions with the environment. Those interactions occur with the managed environment, with a user interacting through a console or with other controllers. Connector places may be source connector places or terminal connector places. The union of source and terminal connector places represents the *interface* of the PTN.

A source connector is a place that interacts with the environment and creates new activity tokens as effect of this interaction. A source connector has only outbound arcs. A

terminal connector is a place that interacts with the environment when it receives an activity token and initiates an action. Activity tokens may be consumed during this interaction. Connector places may exist that are neither source nor terminal to represent intermediate stages. Connector places supplement PTN for DSM and OSM allowing so-called activity tokens to travel.

8.1.6.2 Activity Tokens

While tokens so far have been introduced to represent managed elements, activity tokens represent actions or state changes associated with managed elements. Activity tokens are the only cause of transitions for regular tokens in a DSM or OSM. Activity tokens are used to initiate and control the transitions in DSM and OSM nets. Activity tokens are associated with a specific managed element (and the representing token) to which the interaction applies.

While regular tokens (representing managed elements) can only travel through regular (non-connector) places, activity tokens may travel both under two special transition rules:

- *Join transition (bonding):* an activity token enables a join transition only for the associated (managed element) token and bonds with it during the transition. It remains bonded until it is detached from the regular token.
- *Fork transition (detaching):* if a bonded token arrives at a fork transition where following places include both connector places and non-connector places, activity tokens detach from carrying regular tokens and pass along the arc(s) to the connector place(s), while the regular token passes along the other arc(s) to non-connector places.

8.1.7 Deriving Actions from Desired State Changes

Figure 42 shows the desired state model from Figure 40 supplemented with connector places $sc_{[1,2,3,4]}$ in a state before and after transition $t_{1,2}$ has fired.

An activity token is shown in a source connector place sc_1 before $t_{1,2}$ has fired. This activity token might have been created as effect of a user interaction to change the desired state from [down] to [up]. The occurrence of the activity token in sc_1 enables and fires transition $t_{1,2}$ (under the assumption that the activity is associated with the managed element represented by the token in sd_1).

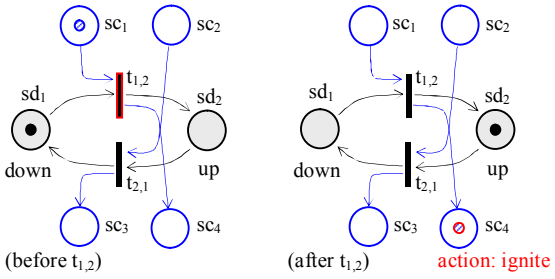


Figure 42: Desired state model before and after firing transition $t_{1,2}$.

(Non-connector places are shown with grey shading in the following figures. Connector places are without shading.)

The join rule applies to transition $t_{1,2}$ (multiple inbound arcs into $t_{1,2}$), which means that the activity token in sc_1 bonds with the one in sd_1. Transition $t_{1,2}$ is also a fork transition (multiple outbound arcs from $t_{1,2}$ leading to connector and non-connector places) such that the bonded tokens immediately separate. The activity token transitions into sc_4 while the token of the managed element transitions into sd_2 (new desired state [up]).

The arrival of the activity token in terminal connector place sc_4 can trigger an action in the managed environment to ignite the boot process. Connector states sc_2 and sc_3 have the reverse effect when desired state is changed from [up] to [down].

Source connector places in DSM such as sc_1 and sc_2 provide the control elements for altering the desired state for a managed element. Terminal connector places in DSM such as sc_3 and sc_4 represent actions that are initiated for a managed element when an activity token arrives.

8.1.8 Reflecting Observed State Changes

Observed state changes as effect of reported changes from the managed environment. The same concept of connector places and activity tokens is applied. Source connector places are associated with sensors in the managed environment creating activity tokens when change is observed for a managed element. Activity tokens bond with the tokens of the managed elements for which changes were observed. Transitions of bonded tokens then lead to changes in the observed state model.

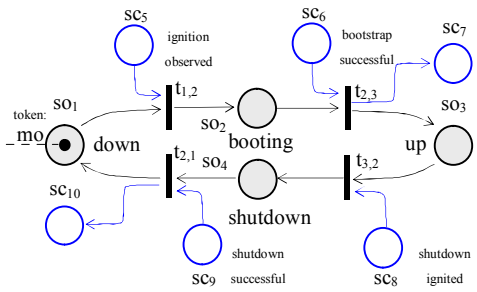


Figure 43: Observed state model with connector places.

Figure 43 shows the observed state model from Figure 40 supplemented with connector places for responding to changes in the managed environment. The token of a managed element resides in place so_1 ([down]). When the boot sequence is initiated (as effect of an activity token arriving in place sc_4 in Figure 42 and triggering ignition), this ignition can be observed in the managed environment and reported as an activity token arriving in sc_5, which enables transition $t_{1,2}$ and, during firing, bonds the activity token to token *mo*. The bonded token then resides in place so_2 ([booting]). When the boot sequence was completed successfully, an activity token is reported to so_6 enabling $t_{2,3}$. Based on the rules for bonded tokens, the token separates from the activity token during $t_{2,3}$, bringing the regular token to so_3 ([up]) and the activity token to sc_7 where it is consumed. Place so_3 is a place that is correlated with place sd_2 in the desired state model. The state of the managed element is now aligned with its desired state.

The nets for DSM and OSM are indirectly connected through connector places. When a terminal connector place in the desired state model receives an activity token, an effect in the managed environment is triggered, which is reported back as another activity token arriving in a connector place in the observed state model. Occurrence of those activity

tokens then can enable transitions in the observed state model. Figure 44 shows this indirect linkage between DSM and OSM. A direct linkage between connector places in DSM and OSM can also be established by connecting places through a direct transition.

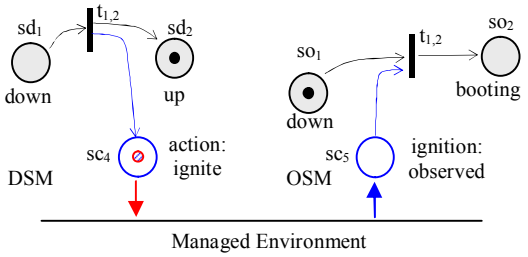


Figure 44: Linkage between initiating change and observing it.

8.1.9 Deriving Actions from Observed State Changes

In addition to intended changes in the managed environment which are derived from changes in the desired state model, error, failures and other conditions may occur in the managed environment any time. When those are reported, they also lead to the creation of activity tokens in connector places. The observed state model must take these conditions into account and must be designed accordingly.

For instance, the boot process of a server may end with a failure or may not complete within an expected time. An additional error place is introduced in the observed state model (so_5 in Figure 45) to reflect those conditions.

The actual status of the server is unknown at this point. As long as the desired state still in [up], the observed state Petri Net may be designed in a way that it includes a sequence of corrective actions by attempting to reboot the server by:

1.) power cycle the server (bringing it into a defined state [down]) and

2.) re-igniting the boot sequence.

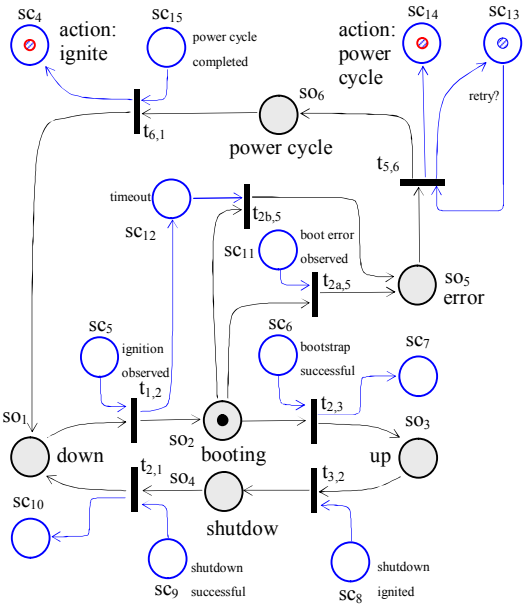


Figure 45: Extended observed state model with error correction.

Figure 45 shows the PTN which handles these cases. Two new places so_5 [error] and so_6 [power cycle] have been introduced as well as five new connector tokens sc_{11} [boot error observed], sc_{12} [timeout], sc_{13} [retry], sc_{14} [initiate power cycle] and sc_{15} [power cycle completed]. The figure shows the token in place so_2 [booting], which has three possible outcomes represented by three transitions $t_{2,3}$, $t_{2a,5}$ and $t_{2b,5}$. They are enabled by activity tokens arriving in sc_6 [bootstrap successful], sc_{11} [boot error observed] or sc_{12} (timeout). Place sc_{12} receives an activity token after $t_{1,2}$ has fired starting the timer as side effect.

In case of error, the token in place so_2 [booting] bonds with an activity token arriving from connector places sc_{11} [boot error observed] or sc_{12} [timeout] and transitions to place so_5 [error]. An activity token in place sc_{13} indicates that the controller should retry the boot cycle leading to the transitions to place so_6 [power cycle]. At this transition, activity tokens are separated and placed into sc_{13} (to maintain the retry marking) and sc_{14}, which is a terminal connector place that starts the power cycle. The end of the power cycle is indicated by an activity token arriving in place sc_{15}, which enables and fires transition $t_{6,1}$. This transition separates the activity token into sc_4 which ignites the bootstrap (see Figure 42). The token now resides in the initial place so_1 [down] and the process repeats.

Whether or not the retry cycle will be performed in case of error depends on the marking of sc_{13}. This marking can be made dependent on whether or not the desired state for the server is still [up] or other conditions (not shown in the figure).

8.1.10 Controller Composition

Multiple controllers will interact in an automated IT management process chain, each responsible for a specific task or managed domain. Coordination among controllers is needed. Higher-ordered controllers mainly perform coordination tasks. They contain the composition models that span across underlying controllers and constitute an automated IT management process chain.

The self-similar structure of the Task Automation Controller allows the composition of controllers as shown in Figure 46. Actions initiated by the upper controller are applied as desired state changes to underlying controllers. And reversely, observed state in underlying controllers constitutes the observed state of a higher-ordered controller. The interaction points among controllers are:

- higher Actuator Connector to lower DSM,
- lower OSM to higher Observer Connector to higher OSM.

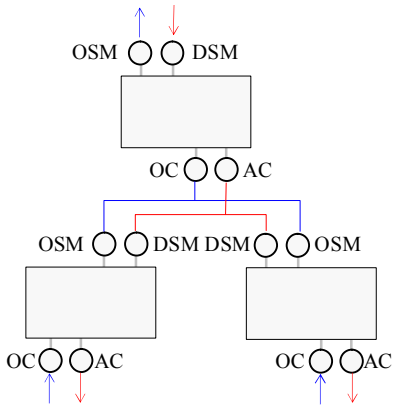


Figure 46: Composition of Task Automation Controllers

Source and terminal connector places are the “interfaces” at the model level. Connector places are accessed through DSM or OSM Interfaces. Activity tokens arriving in terminal connector places can cause inter-controller activity. Activity tokens arriving in source connector tokens can then cause changes in PTN. Events can be issued to subscribers when the OSM or the DSM changes, which also lead to the creation of activity tokens in source connector places in PTN.

All those interactions are mediated through "get", “put”, "transfer", and "push/subscribe" operations defined for web service management standards [Gra04b], [WS-MAN].

8.1.11 PTN Execution Engine

A PTN Execution Engine was implemented (in Java) that interprets DSM and OSM. It forms the core part of the Task Automation Controller. PTN Schemata have been defined for DSM and OSM models which allows to represent PTN models in XML. Places are static XML fragments which can be addressed by xpath expressions. Tokens are represented as dynamic XML fragments that are associated with one place at a time.

The engine operates on the XML (SAX) trees of the DSM and OSM PTN models and interacts with the web service endpoint interfaces provided by the controller toolkit. Operating directly on XML tree representations of models also ensures that model state exposed through the DSM and OSM interfaces is always up to date. The engine is triggered when a new activity token arrives through one of the controller interfaces. Only arrival of activity tokens can alter model state.

Java class names are associated with terminal connector places, which are instantiated when activity tokens arrive. Updates or events associated with DSM or OSM are directly transformed into the creation of activity tokens in the addressed places along with the invocation of the engine.

A toolkit [Gra06] was developed for implementing controllers based on web services management standards. This toolkit was used for building the controllers for the database use cases described later. Controller interfaces employ web service management standards [Gra04b]. Web services management standards WSRF [WS-RF] and WSDM [WS-DM] were initially employed using the open source WSRF implementation from Globus GT4 [GT4] as basis for the toolkit supplemented with WSDM schema. The controller toolkit has been refactored to support the more recent WS-Management [WS-MAN] standard. All web services management standards provide similar operations to access XML representations of models as well as event notifications.

Web services management standards achieve interoperability at the interface level. In regard to models, they only require that models are or can be rendered in XML. They do not impose a specific modeling framework. Interoperability at the model exchange level requires additional agreement. In the current controller realization, models are defined as XML schema and are proprietary. Compliance with the recently emerging modeling framework Service Modeling Language (SML) [SML] is desirable and will be factored into controller models as its common and core model definitions mature.

8.1.12 Controller Automation Use Cases

This section describes a controller-based automation scenario that was built as joint effort between a team from HP and one from Oracle using the controller toolkit. The goal was

to demonstrate model-driven automation for selected task automation use cases. The testbed consisted of HP blade servers (eight servers of type BL20p, dual Pentium III, 3.2GHz, 4GB), HP SAN disk array with two fiber channel switches, and a HP ProCurve 2848 LAN network.

The automation use cases demonstrated coordinated lifecycle and auto-correction capabilities for error situations that would require human attention during operation in a traditional system. The three automation use cases were:

1.) Automated provisioning and coordinated lifecycle control of an Oracle database on blade servers.

2.) A storage auto-correction capability by automatically configuring and attaching new disks from the SAN to servers when Oracle Enterprise Manager predicted storage shortage due to growing table sizes in the database.

3.) Response-time auto-correction during operation by flexing additional blades into the database pool when response times increased above a threshold due to load increases.

All use cases required the direct interaction between the two management systems from HP and Oracle. Neither system could achieve them alone. All interactions between systems were normalized as model exchanges between controllers.

To actuate actual changes in the managed environment, two management systems were employed: HP blade server automation software and Oracle Enterprise Manager. Both systems had to cooperate in order to solve the automation use cases, which would have required the point integration between the two systems in a traditional approach. Instead, both systems were wrapped into controllers using the controller toolkit.

Three controllers were created with models:

- HP Blade Automation Controller,
- Oracle Enterprise Manager Controller, and an additional
- Coordinator Controller.

The first two controllers were implemented as wrappers around HP blade automation infrastructure and Enterprise Manager from Oracle. The third controller was created for coordinating the two other controllers. It coordinated activities and composed them into one management service achieving all three use cases.

The first use case allowed for basic provisioning of blade servers, disks and networks using HP blade automation infrastructure. It deployed Oracle 10g on bare servers and configured it for management through Oracle Enterprise Manager. Two template models were available for instantiating the database representing three different configurations ("sizes") of the Oracle database deployment: small (one blade server), medium (two servers) and large (4 servers). Templates were chosen from the coordinator controller based on user input. This specification was based on the number of users, data set size and the transaction rate supported by a configuration. After template selection, the coordinator controller interacted with the underlying controllers to establish the needed hardware infrastructure using the capabilities of HP's blade server automation software and, once this had been achieved, to initiate the configuration of Oracle.

The second use case employed Oracle Enterprise Manager's ability to predict shortage of storage in a growing database and triggering correction by notifying the HP Blade Automation Controller to attach another disk from the SAN storage array. After completion, the Oracle controller was initiated to reconfigure the database in order to utilize the additional disk.

The third use case allowed to auto-correct server capacity when slowing database response time was indicated by Oracle Enterprise Manager, triggering a server flex-up operation to the HP Blade Automation Controller. After completion, database instances configured into the database.

These use cases demonstrated automation scenarios achieving self-correcting behavior. They also demonstrated how controllers can be used to wrap legacy management systems into a controller framework, normalizing their interactions using common controller interfaces and model exchange through interfaces. Use of PTN allowed describing and executing the coordination needed between controllers providing an example for an automated IT management process chain.

8.1.13 Related Approaches

While automation has made substantial progress on the business side of IT, such as in business process automation [Sche04], automation in IT management has been lagging behind. On the business side of IT, enterprise software such SAP is widely been used to automate the processing and management of enterprise information. Tools such as ARIS [IDS] are used to design automated business processes. In IT management, in contrast, people still carry out management processes from higher-ordered planning stages to the lowest levels of managing machines, networks and storage. Management tools are used that support those tasks. Tools signal and report conditions to a human operator, who then is in charge to interpret those signals and eventually respond by making a change in the system, which is again mediated through a tool. The loop is not closed in IT management; the operator's attention is permanently required.

Workflow systems are predominantly used in IT task automation. In contrast to workflow languages, which describe sequences of parallel or sequential actions, Petri Nets primarily represent state. State changes occur in effect of transitions, which are also described in a Petri Net. State in a workflow language is always external to the actual workflow description. It typically occurs in form of a message that is processed along the workflow statements (e.g. a purchase order, which is the message, traveling through a purchase order workflow, which is a graph of actions).

Cfengine was developed at University College in Oslo [Burg93]. Its primary function is to provide automated configuration and maintenance of computers, from a policy specification. It emerged from the need to control the accumulation of complex shell scripts used in the automation of key system maintenance. In a heterogeneous environment, shell scripts are hard to maintain: shell commands have differing syntax across different operating systems, and the locations and names of key files differ. The non-uniformity of Unix was a major problem. Cfengine defined a new language which unified the heterogeneity underneath. The aim was to absorb frequently used coding paradigms into a declarative, domain-specific language that would offer self-

documenting configuration. Cfengine has an agent-based infrastructure through which scripts can be distributed and executed on machines.

While Cfengine allows to abstract and to unify scripts used in system management, it does not posses the capabilities of a controller. Cfengine needs to be activated by an administrator in order to perform management tasks on remote systems.

8.1.14 Summary

The section presented a Task Automation Controller which adopts the concept of a feedback system to automated IT management for operational tasks such as lifecycle management. State of a managed environment is represented in terms of a pair of models for desired and observed states and transitions between those states. A specialized form of a Place-Transition-Net (PTN or Petri Net) is used to represent the static aspects (states) as well as dynamic aspects (coordination) in one model. This overcomes the problems that result from separating models from interpretation and execution logic that is often found in model-based management approaches.

A PTN execution engine was built that directly executes PTN and forms the core of the controller logic. This allows the controller logic to be "generic" and driven by configurable PTN models as opposed to hard-coded and built into the controller. Web services management standards are used to create uniform interfaces to controllers and that allow the composition of controllers. It was shown how error correction can be factored into PTN. It was also shown how coordination between controllers in an automated IT management process chain can be achieved. Three automation tasks for deploying the software configurations for a database system onto server resources have been implemented using the controller.

8.2 The Deployment Manager

The Deployment Manager relies on an existing deployment infrastructure which is called Radia [RAD]. Radia is a deployment system that originally has been developed by Novadigm, Inc., which has been acquired by Hewlett-Packard in 2004.

The Deployment Manager also represents the approach of integrating an *existing* management system into the architecture of the DCI-OS as a module, which is essential since not all functionality can be developed from ground up for the DCI-OS.

Integration and integration capabilities for existing management systems are thus essential for a DCI-OS. Two major aspects must be provided for integration into a DCI-OS environment:

- A programmable interface through which the DCI-OS interacts with the system,
- An information representation and mapping bridging the gap between the information models used in the DCI-OS and the information models used by the management system.
- A standard's based middleware providing the instantaneous exchange of information between the DCI-OS and connected management systems.

8.2.1 The Radia Deployment Manager as Integration Example

Like other management systems, Radia relies on an administrator role for configuring resources and deploying configurations. An administrator (a person) sets certain policies in the central Radia configuration server which are then "rolled out" to the connected sets of resources automatically in a controllable batch modus.

Since Radia already provided deployment automation, the task here was to enable programmability for the initial configuration setting on the central deployment server, which before was only possible by an administrator. In order to achieve this task, interfaces needed to be developed providing programmatic access to the central Radia management server. A specific task was the transformation of the CIM-based models driving the operations of the DCI-OS into the specific, proprietary format Radia used in its internal information model.

Radia followed a declarative paradigm of constantly comparing Desired State (the state defined for a resource on the central Radia management server) with Observed State (the state constantly being reported from the associated resource) deriving actions from the differences. This close proximity of the operational models simplified the integration of the Radia Deployment Manager into the DCI-OS significantly.

The Radia Deployment Manager includes an abstracted, externally exposed set of desired and observed state models upon which an interface is constructed based on web services management standards providing operations on model states, which in turn drive changes in the underlying resources utilizing Radia's capabilities. This approach did not require changing Radia itself, which was a significant advantage.

Figure 47 shows the integration diagram of the Radia Deployment Manager into the DCI-OS. The adapter shown in the upper left part of the diagram connects the Radia system with the DCI-OS Infrastructure Services layer.

Input into the Radia Deployment Manager are settings for the Desired State model maintained in the central Radia management server and changes to this model.

The lower part of the diagram shows the Radia system connected to machine resources and maintaining certain software configurations on those machines as expressed in the Desired States defined for those machine resources. A Radia agent exists on each system requesting the information about what has been set as the Desired State of that system and compares it to its current state. In case of difference, the agent per-forms actions to transition this state toward the Desired State by installing, upgrading or removing packages. Desired State definitions for classes of systems or users, which are subsequently referred to as “Entitled Entities”, are defined in the model in the central Radia Configuration Server, which is Radia's central configuration server. Radia uses internally a model of an object-oriented database (OO-DB). The Radia administrator uses a UI tool, the Radia System Explorer, to set Desired State definitions in RCS. For the integration with a DCI-OS, a second channel was created through the Radia Adapter to set the Desired State programmatically from outside Radia based on a model which has been defined for Radia deployments.

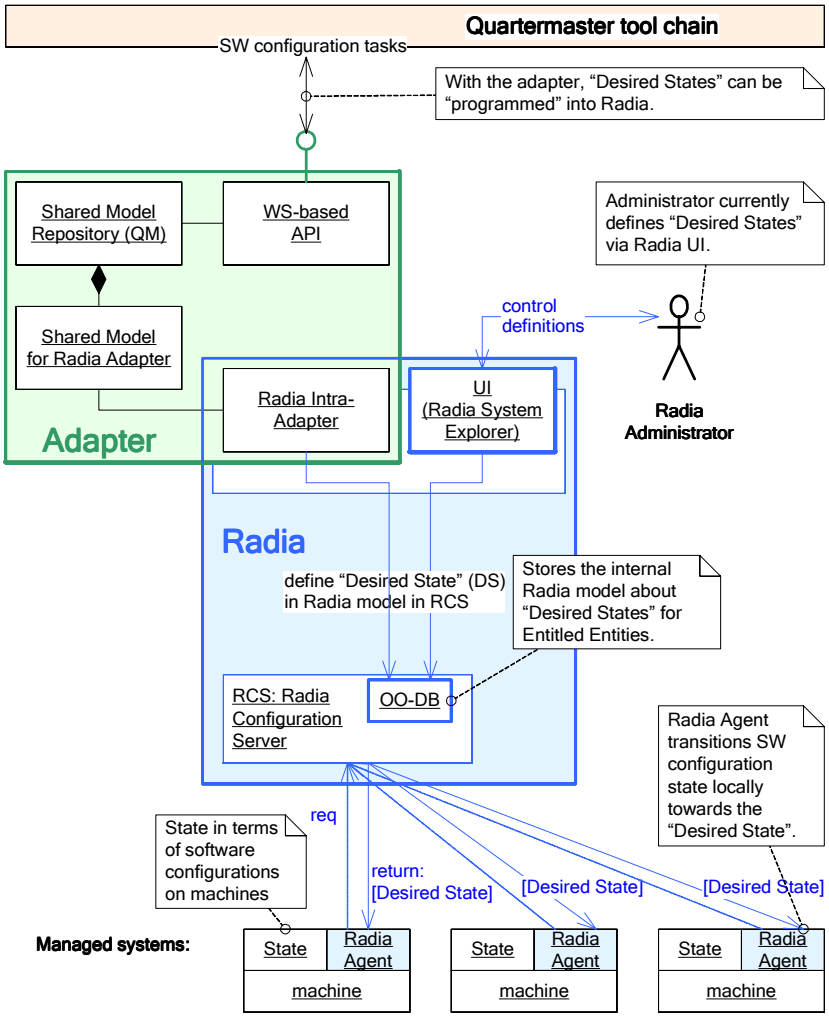


Figure 47: Integration of the Radia Deployment Manager into the DCI-OS.

8.2.2 External Integration Model

The adapter for the Deployment Manager differentiates between its internal models, which refer to the detailed information each system needs to maintain for its own operation, and external or shared models, which contains only the information needed for the external interaction with a system. Internal models are hidden by the deployment adapter. External models are exposed and accessible through the interface with the DCI-OS. Figure 48 shows the external model which has been defined for the Radia Deployment Manager based on three abstractions: REEntity – the resource to which a configuration can be applied; RServices – the configuration that can be applied; and REntitlement – an association between REEntity and RService when a configuration should be applied. The interactions with the DCI-OS are driven by performing operations over this externally shared model.

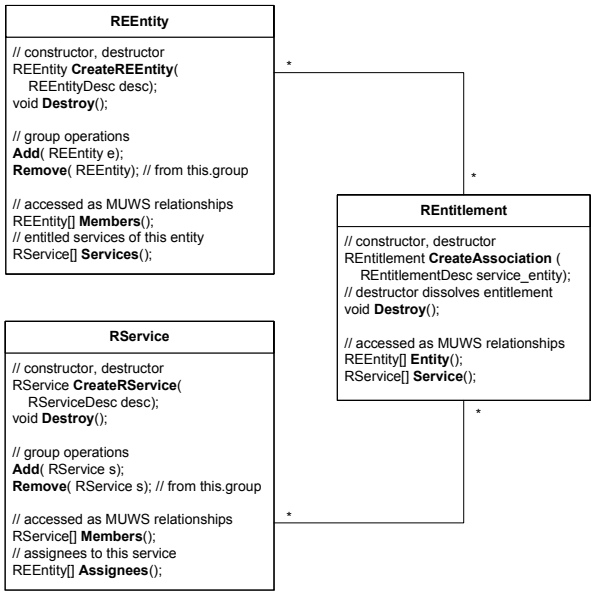


Figure 48: Externally shared integration model for the Radia Deployment Manager.

The concepts occurring in the external model representation translate into the following model concepts used by the Radia Deployment Manager internally:

- REEntity – Radia Entitled Entity (a system, a user, or groupings of those such as departments, regions, etc.),
- REEntityDesc – Radia Entitled Entity Description based on REEntity meta-data,
- RService – Radia Service (a service instance that can be entitled to REEntities),
- RServiceDesc – Radia Service Description based on RService meta-data,
- REntitlement – an entitlement between entity and service,
- REntitlementDesc – the description of the entitlement association.

The model representation in the Information Model Layer of the DCI-OS are based on CIM classes. CIM representations must be developed for the Radia Deployment Manager for entity classes (RService, REntitledEntity) and association classes (REntitlement).

Classes are defined as .mof and incorporated into the CIM repository by creating class instances. Instances of classes (such as for specific services, entitled entities, or entitlement associations) can also be defined in CIM's Management Object Format (MOF) or can be incorporated from a CIM instance provider. During run-time, class and instance information can be queried and manipulated via the API. Changes in the Shared Model then drive change in the Radia infrastructure.

Figure 49 shows the CIM framework of the DCI-OS used to represent of the externally shared model as used in the DCI-OS.

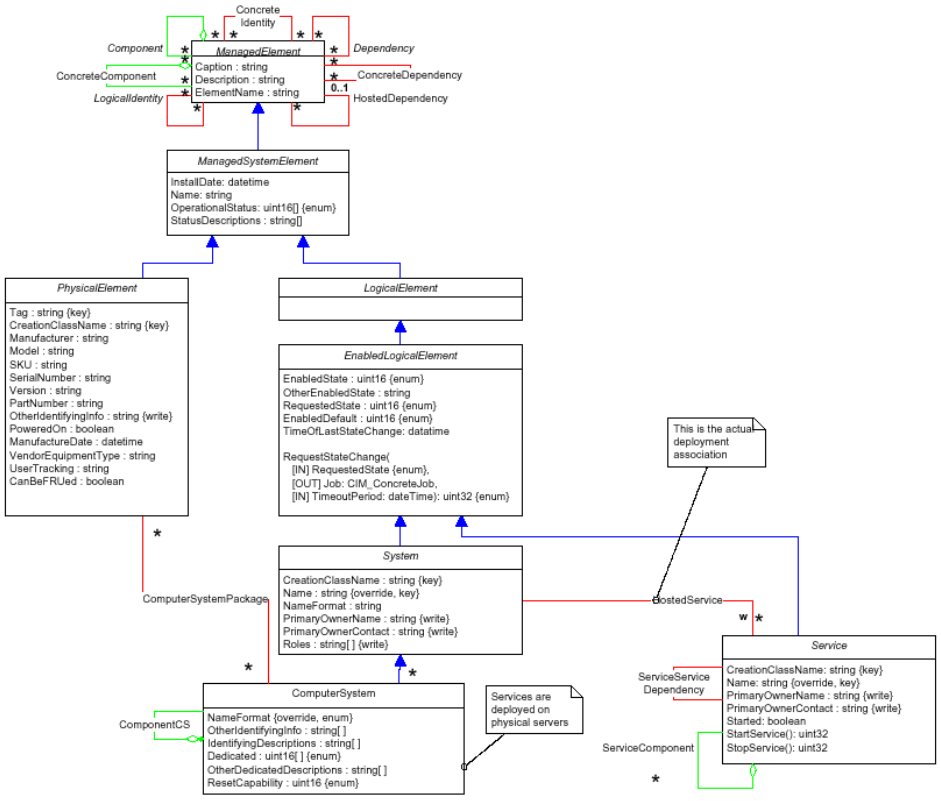


Figure 49: CIM representation of the integration model as used in the DCI-OS.

8.2.3 Standard's-based Web-Services Middleware

A machine-accessible interface is needed for the Radia deployment adapter. In general, a deployment adapter interface requires:

- an event mechanism allowing external entities to subscribe to events such as model change events,
- support for interpretation of delivered information (type information, schemas, semantic information), and
- control operations that can be invoked in order to trigger action in the underlying system.

Web services management standards meet the requirements for deployment adapter and should thus form the basis for deployment adapter interfaces.

A number of web services based management standards have been proposed over the years, beginning with WBEM, an HTTP/XML-based model exchange protocol for CIM. WBEM has been developed and standardized by the Distributed Management Task Force (DMTF) [DMTF]. WBEM has significant weaknesses in terms of event capabilities, which were overcome by middleware frameworks such as proposed by the Global Grid Forum (GGF), later the Open Grid Forum (OGF) [OGF] as standards for Grids, standards which now have been merged into OASIS [OASIS]. These standards were the Web-Services Resource Framework (WS-RF) [WS-RF] and the Web-Services Distributed Management (WS-DM) [WS-DM]. [Gra04b] shows the relevance of these middleware

standards for management system integration in the enterprise and data center contexts since they allow standardized web-services-based integration of enterprise applications as well as their associated management systems. Implementations of those standards exist in form of the Globus Toolkit [GT4].

The OASIS Web Services Distributed Management (WSDM) [WS-DM] working group continued with the MUWS (Management Using Web Services) standard [MU-WS] with participation of HP that is based on the Web Service Resource Framework [WS-RF]. MUWS is the basis for the deployment adapter interface.

In the case of the Radia Deployment Manager, REEntities, RServices and REntitlements are considered Web Services Resources at the web-service interface layer according to the WS-RF specifications. Each has a Web Services Resources class associated and exposes one Web Services Endpoint (WSE), which is an entity which is addressable in the web services middleware through a URL. Access to different class instances is differentiated by Reference Properties as defined in the WS-Addressing web services standard.

Each instance and the class (accessing the endpoint without reference properties) can act as factory for creating a new instance. To create the instance, a corresponding description is passed (REEntityDesc, RServiceDesc or REntitle-mentDesc). The factory methods return a WS-Adres sing endpoint reference that consists of the endpoint providing access to the Web Service representing the class and reference properties to identify the instance. The MUWS relationship mechanism is used to represent the linkages for the entitlement association REntitlement between REEntity and RService. A REntitlement instance has a relationship link to the respective REEntity and RService instances (type 'urn:radia.hp.com/reentity/ rservice'). REntity and RService instances in turn have relationships with respective REntitlement instances (types 'urn:radia.hp.com/reentity/rentitlement' and 'urn:radia.hp.com/rservice/rentitlement'). All these relation-ships are automatically established using the CreateAssociation operation on a REntitlement instance. Relationships are also used to represent the REEntity and RService group membership (types 'urn:radia.hp.com/reentity/reentity' and 'urn:radia.hp.com/rservice/rservice'). The Add/Remove operations create or destroy a relationship. Each of the three Web Service interfaces expose all WS-ResourceProperty operations (which can be used to access the description and relationships) and WS-Notification operations. A client can subscribe to notifications for Resource Property changes. Since MUWS relationships are accessed as Resource Properties, the client can receive notifications for relationship changes.

The Radia Deployment Manager exposes its operations through this web-services based interface providing operations to set and reset states in the exposed Desired State Model.

Next, the Resource Acquisition Manager is discussed.

8.3 The Resource Acquisition Manager

The Resource Acquisition Manager forms another important part of the Infrastructure Services Layer. Its task is to provide the access to the data center's resources via communicating with the DCI-OS Resource Management Layer, which is discussed in the next chapter in more detail.

The Resource Acquisition Manager also follows a controller pattern where a certain level of resource supply is constantly compared with the actual resource usage consumed by the Infrastructure Service [Gra03a], [Gra04c]. Based on the comparison and depending on resource types, resources can be acquired in addition to the current level from the DCI-OS Resource Management Layer or released back. The organization of resources in form of pools allows that resources can be acquired and released depending only on type and capacity, but not depending on particular instances. For instance, when a server resource is required, it means that any instance of a particular server type with a certain capacity can be assigned. The assigned server resource is then provided with specific configurations in the deployment step from the context of the Infrastructure Service into which it was assigned. This principle of isolating resources from configurations enables the management of resources as pools. It allows more flexibility at a cost of an additional deployment step. However, this step is automated by the deployment managers and not manual as it is the case in today's practice.

The process of adjusting the amount or number of resources according to demand in an Infrastructure Service instance is called *flexing* [Gra04d].

The following discussion shows flexing for the resource type of servers in context of a horizontally scalable enterprise application. *Server Flexing* is to the process of adjusting numbers of servers with instances of an application running on them depending on current workload conditions. Adding servers with instances running on them expands the capacity of the application for serving higher workload. Removing servers reduces capacity for meeting lower demand. This principle applies to so-called *horizontally scalable applications*. Horizontally-scalable applications are simultaneously operated on a number of servers of same type. Examples are web servers, application servers or clustered databases.

The control system of the Resource Acquisition Manager automatically performs the control loop over the stages of monitoring, assessment, and corrective action following a goal. A goal, for example, can be to keep the operational parameters of a controlled system within defined ranges. Capacity of server groups is automatically adjusted during operation by acquiring or releasing servers.

In this discussion, the control system of the Resource Acquisition Manager interacts with the Resource Management Layer of the DCI-OS via a standard web services protocol defined by the Open Grid Services Infrastructure (OGSI) middleware standard [OGSI] leveraging OGSI's built-in security and event models. More about web services-based protocols used as communication middleware can be found in [Gra04b] and [Gra06].

8.3.1 Resource Group Flexing

We refer to servers as resources in this section. A server group consists of a number of servers of same type, running the same version of the operating system and application, both originating from the same disk image. Life-cycle scripts are executed when the operating system is booting for the final configuration (see Figure 54). Each server uses its own copy of a master image. The process of creating that copy is in the range of minutes limiting the periodicity of the control loop to 10's of minutes or hours, which is adequate to accommodate longer-term daily, weekly or monthly patterns in commercial workloads.

Reasons for flexing the server number up might be that application load is increasing and servers are approaching saturation levels. Reasons for releasing servers from server groups might be that servers are needed for other applications. Servers operated under a pay-per-use regime provide incentives to farms and their owners to free resources when they are not needed.

Server flexing can occur:

- Statically, by stopping the entire application environment, performing all necessary reconfigurations for joining or releasing servers to or from a server group, and restarting the application environment with the new configuration. Static flexing is typically performed manually today.
- Dynamically, by joining in new servers into a server group of an operating farm and gracefully releasing them from an operating farm. Dynamic flexing relies on automatable reconfiguration capabilities supported by virtualized resource environments.

8.3.2 Resource Flex Control Loop

In a server flex control loop, flex operations are initiated by a control system that continuously oversees the conditions in the farm (based on load, use patterns, failure conditions, etc.) and evaluates conditions in order to come to a conclusion to increase or decrease the number of servers in a server group. Figure 50 shows a control loop for server flexing. The control loop consists of three main stages monitoring, assessment, and adjustment.

Monitoring. – Monitoring is typically based on elementary resource metrics such as CPU utilization, process queue lengths, used network bandwidth, memory utilization and swap rate. If application instrumentation is provided, application-level metrics can be taken into account as well such as transaction rates, response times, numbers of simultaneous sessions, etc.

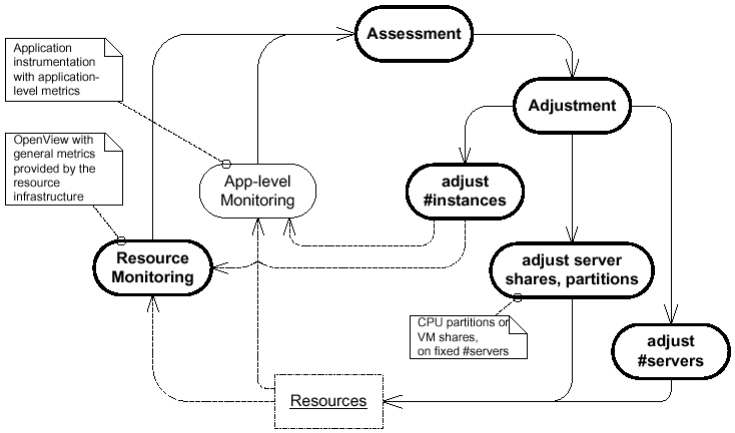


Figure 50: Control loop for server resource flexing.

Assessment. – Given measurements as input, the conditions in a server group can be assessed whether the system is considered operating within defined bounds or not. Automating assessment can be simple such as observing thresholds or it can be complex. Policy defines the behavior of the assessment system. Policy must be customizable.

Adjustment. – As a result of assessment, a decision about a corrective action is derived, which leads to an adjustment in the system. Various means for adjustments exist for horizontally-scalable applications:

1. The number of instances of an application can be increased or decreased.
2. On certain hardware, CPUs or CPU partitions can be added or removed for an application in order to adjust processing capacity. Virtual machines allow adjusting CPU shares between virtual machines.
3. The number of physical machines can be adjusted (constant in the first two cases).

Virtualized resource infrastructures have the capability to dynamically (during the operation of an application) add or remove physical servers from an application environment. It has the capability to configure servers into that environment, or release and unconfigure them from the environment.

The outer-most control loop in Figure 50 encompasses the acquisition and release of entire physical servers from the resource infrastructure.

The three dimensions of adjustments also reflect different levels of granularity in terms of resources (CPU cycles, CPU partitions or CPUs, or entire servers) and time scales of shorter-, mid-, or longer-term control loops, which may co-exist. When a finer-grained loop has reached its limits, the control loop of the next level can expand its reach.

8.3.3 Adaptive Control System for Resource Group Flexing

The purpose of the adaptive control system for server group flexing is to provide horizontally scalable applications the ability to acquire or release physical server resources using the capabilities of the Utility Data Center to configure and unconfigure servers from the virtualized resource environment of a farm.

8.3.3.1 Requirements for Flexible Resource Acquisition and Release

A number of requirements have been considered for the definition of the control system:

- *Automation* driven by policy. – The control system operates without involvement of the operator after the operator has set the control policy.
- *Application focus.* – The control system, specifically the assessment and the decision-making part, must be customizable for specific applications and operating regimes defined for a data center.
- *Open-standard's-based interface* and ability to integrate with application management and control systems. – Applications are built more and more for utility modes of operation. It means that they incorporate interfaces and components for their management and control. Application vendors enrich their applications with such management systems. One major requirement of the adaptive control system here was being able to integrate with those application management systems providing them the link, based on open standards, into the resource infrastructure where they can acquire or release configured servers.
- *"Plug-and-Play".* – Application integrators or data center operators must be able to define control loops easily. A "pluggable" component design based on open standards should form the foundation.

- *Trust and Security.* – Control functions in data centers are hard to automate not only because of technical complexity. Automating tasks relies on trust which operators must develop in order to delegate functions they perform manually today to automated systems. Security breaches and malfunction are main concerns that must be prevented by automated systems.
- *Integration into higher-ordered control systems.* – The control system discussed here addresses the rather limited scope of flexing individual server groups for horizontally-scalable applications. Higher-ordered control systems can expand the scope to entire farms, farm groups, data centers, or even across data centers. Those scenarios have been considered for the design, but are not discussed here.

8.3.3.2 Design Choices

The following design choices are guided by requirements:

- Split of the control loop into a fixed infrastructure part embedded in the resource infrastructure providing server configuration, basic resource monitoring, and a customizable Control Plug-in containing the assessment and adjustment functions.
- Defining an open-standard's-based interface between the Control Plug-in and the resource infrastructure.
- Selecting OGSI as interface and interconnect technology mainly because of openness, web services standards, and built-in security and event models.
- Simple interaction patterns between the Control Plug-in and the underlying resource infrastructure based on service invocations and events.

The following figure shows the split of the control loop into Control Plug-in and resource infrastructure.

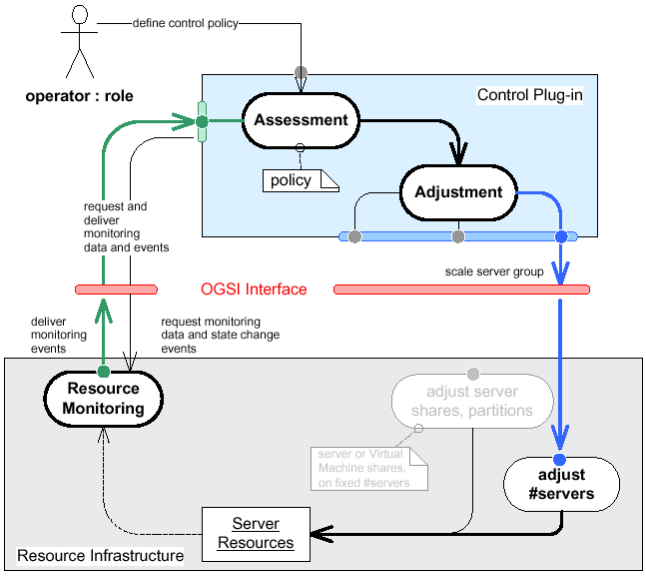


Figure 51: Component view of control loop.

The OGSI interface between Control Plug-in and resource infrastructure primarily has two main functions:

- Request and delivery of basic resource monitoring data (such as provided by HP OpenView [HPOV]) and delivery of state change events such as a change in the number of servers currently participating in a server group.
- Request to scale a server group up or down to a desired target number of servers.

Control policy is defined by the operator or the application administrator, in the current system in form of simple thresholds that are parameterizable for Control Plug-ins, which are implemented as Java OGSI clients.

8.3.4 Resource Flex Control

Unassigned servers are maintained in infrastructure resource pool. When the Control Plug-in initiates scaling a server group up or down, selected server resources transition through the stages shown in Figure 52.

8.3.4.1 Flex up Cycle

A server (or a number of servers) matching the type of the server group is selected from the resource pool and set into the join state. During that stage, the farm resource environment is reconfigured to include the new server(s). This means that the VLAN is reconfigured, virtual IP addresses are assigned to the server, DNS is reconfigured, and the logical disks from the storage array with copies of the server group images are created and mapped onto the joining servers. After that, the joining servers reboot and launch applications from those disks. The join state is finished when new server(s) have booted and applications are launched. Launching applications is initiated by start-up scripts configured on images.

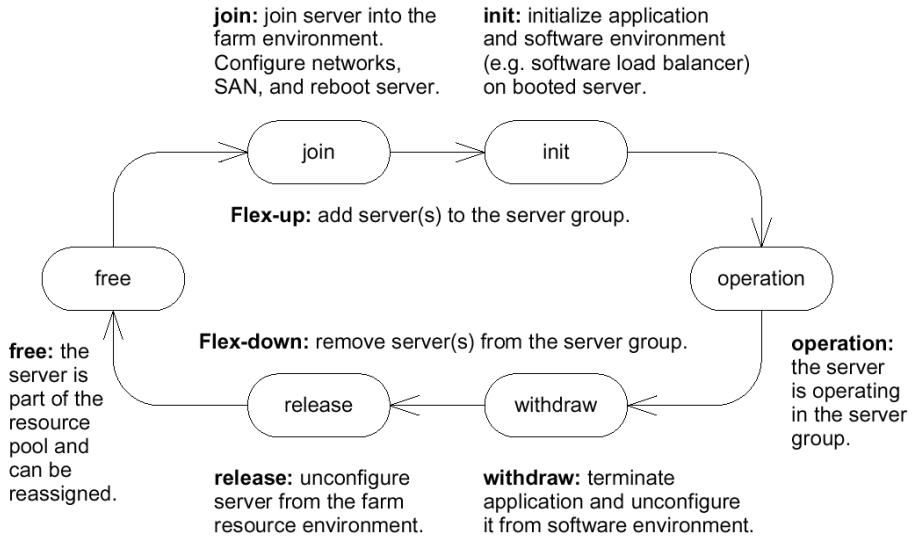


Figure 52: State diagram for server flex operations.

The following init state allows executing further life cycle operations on joined servers (shown as ALCD – Application Life Cycle Daemon in the interaction diagram in Figure 54). This allows reconfiguration in the software environment, for instance, to start

directing workload to joined servers. At the end of the init state, the new servers are fully operational as part of the server group.

8.3.4.2 Flex down Cycle

The flex down cycle is also initiated by the Control Plug-in by selecting a number of servers to be released. In reverse order to flex-up, life cycle scripts on servers are executed first during withdraw, unconfiguring applications of affected servers from their environment. A load balancer may need to be reconfigured to not send further workload to affected servers; active sessions have to be finished, etc. At the end of the withdraw state, the local application instances are no longer part of the application and can finally be terminated.

During the release state, the servers to be released are unconfigured from the farm's resource environment including the VLAN. Disks (logical volumes) are detached by reprogramming the SAN and cleared. No state of application instances is kept for server groups.

At the end of the release state, all state associated with affected servers has been removed from the software and hardware environment including the servers themselves. Servers now transition to the free state and return to the server pool for new assignment.

8.3.4.3 Resource Flex Protocol

The server flex protocol encompasses two parts. Monitoring and state change events must be delivered to the Control Plug-in and eventually other subscribers. And flex instructions (scale server group requests) must be delivered in opposite direction.

Event delivery. – Event delivery is based on OGSI events, which are based on changes in Service Data Elements (SDE) [OGSI]. SDEs are maintained in Interface Instances. Two SDEs have been defined.

SDE SN – the current number of servers (SN – Server Number) in a server group. This SDE triggers events informing the associated Control Plug-in and other subscribers when either the farm administrator has reconfigured the server group through a console, or the server number has changed in effect of a previous flex operation. If the SN reported in the SDE equals to the target SN previously issued as flex request, the Control Plug-in concludes that the previous operation has succeeded. SN change events are trigged at state changes init->operation and release->free.

SDE LL – represents the current Load Level (LL) in a server group. This is an aggregate number taking various OpenView base measurements into account: CPU load, memory usage, IO and SWAP activity. Aggregation is based on a model that for any server i in a server group LLi = max(CPU, MEM, IO, SWAP), with each of the base measurements normalized to a scale of 0-100. The LL of the server group then is LL= Σ LLi. This simple model assumes that load between servers in the server group is evenly distributed.

The rate of LL SDE change events is controlled by OGSI's event flow rate control mechanism with min_interval=30sec and max_interval=300sec.

Based on SN and LL change events, which are also time stamped at the source, the Control Plug-in can obtain a load and utilization profile over time and assess changes in the server group that may lead to flex decisions. Policy parameters in the Control Plug-in must be tuned to avoid thrashing effects.

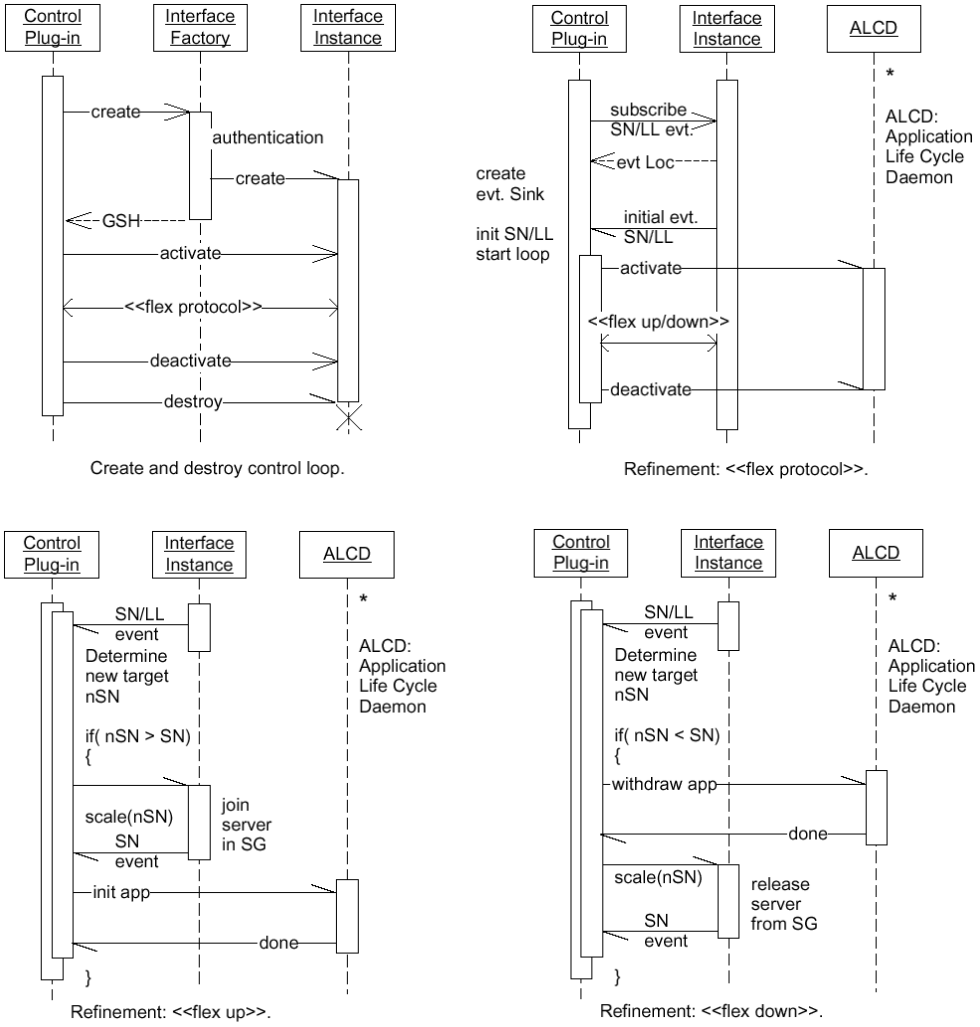


Figure 53: Interaction diagrams for server flex operations.

Flex request. – Once the Control Plug-in has come to the conclusion to flex a server group, it issues a scale server request and waits for its completion indicated by a SN change event. Since execution of flex operations is in the range of 10+x minutes, flex requests are asynchronous. The Control Plug-in can observe the time within which it expects a flex operations to complete and take action when this time passes without receiving a server number change event.

Flex requests are issued with the total target number of servers rather then incremental changes in order to make flex requests idempotent. Failure or incompletion within the expected time frame simply may cause the Control Plug-in to repeat the request leading to the same result.

Figure 54 shows the effect of the server flex control system with the absolute server group load level decreasing over time from about 300 to 0. Accordingly, the Control Plug- in reduces the number of servers in the server group from 4 to 1, leading to the flat server utilization curve shown in the diagram.

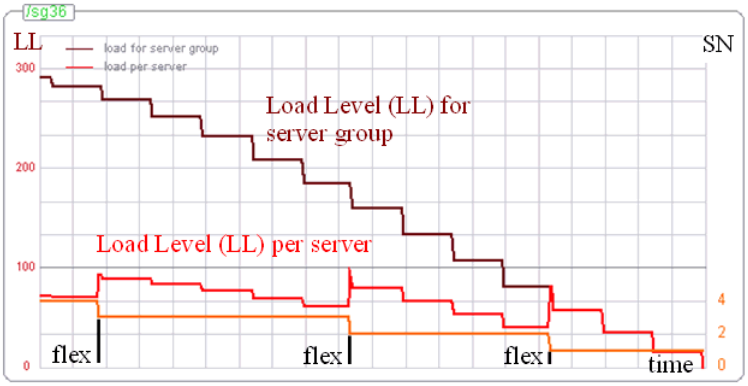


Figure 54: Effects of flexing on workload.

8.4 Summary

Role of the Infrastructure Services layer is to provide the execution environments for Application Services. The Infrastructure Services Layer provides support systems which allow acquiring and releasing resources from the data center-part of the DCI-OS as well as deploying configurations onto those resources fitting the needs of the Infrastructure Service. The tasks of this layer are performed by three main modules: the resource *Acquisition Manager* – allowing the Infrastructure Service to request and release resources of specified types and quantities; the *Deployment Manager* – allowing specific configurations to be deployed onto freshly acquired resources; and the *Task Automation Controller* – coordinating the various activities which have to occur when new resources are provisioned as well as coordinating the overall lifecycle control tasks of the Infrastructure Service (such as create, start, suspend, resume, stop, destroy).

The purpose of the Infrastructure Services layer is similar to the purpose of a run-time system in operating systems. While an operating system run-time library also allows acquiring and releasing resources from the underlying operating system, the task spectrum in the Infrastructure Services layer is much broader. In an operating system, resources considered are simple resources such as memory, processor shares or external storage. For Infrastructure Services, resources encompass entire machines, storage devices, and networks requiring additional application of service-specific configurations.

Unlike in an operating system where the run-time system is part of the same process it controls, the modules of the Infrastructure Services layer execute on *separate management machines* which are dedicated to the DCI-OS. Reason for this decision is to avoid interference between resources (e.g. machines) where Application Services execute and where management functions execute.

Three specific modules have been presented from the Infrastructure Services layer:

- the Task Automation Controller,
- the Deployment Manager, and
- the Resource Acquisition Manager.

The case of the *Task Automation Controller* has been discussed because it demonstrates a principle of declarative control over coordination tasks, which has been found beneficial over traditional workflow-based coordination implementation (e.g. with workflow or business process execution engines or script executions). The declarative approach follows the pattern of a *Controller* which allows setting a state a Desired State (the declaration of *what* should be) as and the controller providing the logic *how* this state is achieved and maintained internally hidden from the outside. Using the declarative approach of a controller is one of the key insights of the development of the DCI-OS

The case of the *Deployment Manager* was presented to demonstrate the integration of an existing management system into the architecture of a DCI-OS for a specific role and a specific purpose. It cannot be assumed that all functionality a DCI-OS requires will be redeveloped from ground up. Rather, a DCI-OS will act as an integration platform of the various tools and systems which are in existence today bringing their capabilities together for achieving deep, integrated automation capabilities which cannot be achieved by individual management systems operated in isolation.

The *Resource Acquisition Manager* establishes the link to the data center resources managed by the DCI-OS by communicating with the Resource Management Layer of the DCI-OS. The Resource Acquisition Manager follows a controller pattern where a certain level of resource supply is constantly compared with the resource usage consumed by the Infrastructure Service. Based on the comparison and depending on resource types, resources can be acquired in addition to the current level from the DCI-OS Resource Management Layer or released back. In this chapter, the case of the Resource Acquisition Manager was discussed by the example of dynamic supply of server resources into an Infrastructure Service for a horizontally scalable enterprise application.

During the discussion of the three components of the Infrastructure Services Layer, aspects of integration between a DCI-OS and existing management systems have been discussed as well as the interface between the Infrastructure Services-related part of the DCI-OS and the Data Center-related part, which both relate to questions around management middleware connecting the various components within a DCI-OS as well as connecting the DCI-OS with other, existing management systems that are used in a particular data center environment.

Three main aspects are relevant:

- The existence or creation of a programmable interface through which the DCI-OS interacts with the system,
- A shared information representation and mapping which needs to be developed bridging the gap between the information models used in the DCI-OS and the information models used by the management system.
- A standard's based middleware providing the instantaneous exchange of information between the DCI-OS and connected management systems.

Both, using a declarative approach to control in the DCI-OS as well as providing the ability to integrate existing management systems into the architecture and the system of a DCI-OS have been key insights resulting from research and development of a DCI-OS. Another innovation insight resulted from separating resource configurations from resource instances. If resource instances are treated independently from configurations needed by specific Infrastructure Services, resource can be managed in pools

interchangeably providing more flexibility. Flexibility comes at an additional cost of a configuration step, which in the context of the DCI-OS is automated.

Chapter 9

The DCI-OS Layer

The DCI-OS layer comprises the lower part in the architecture of a DCI-OS. It is closest to the resources in a data center. Its components are shown in Figure 37.

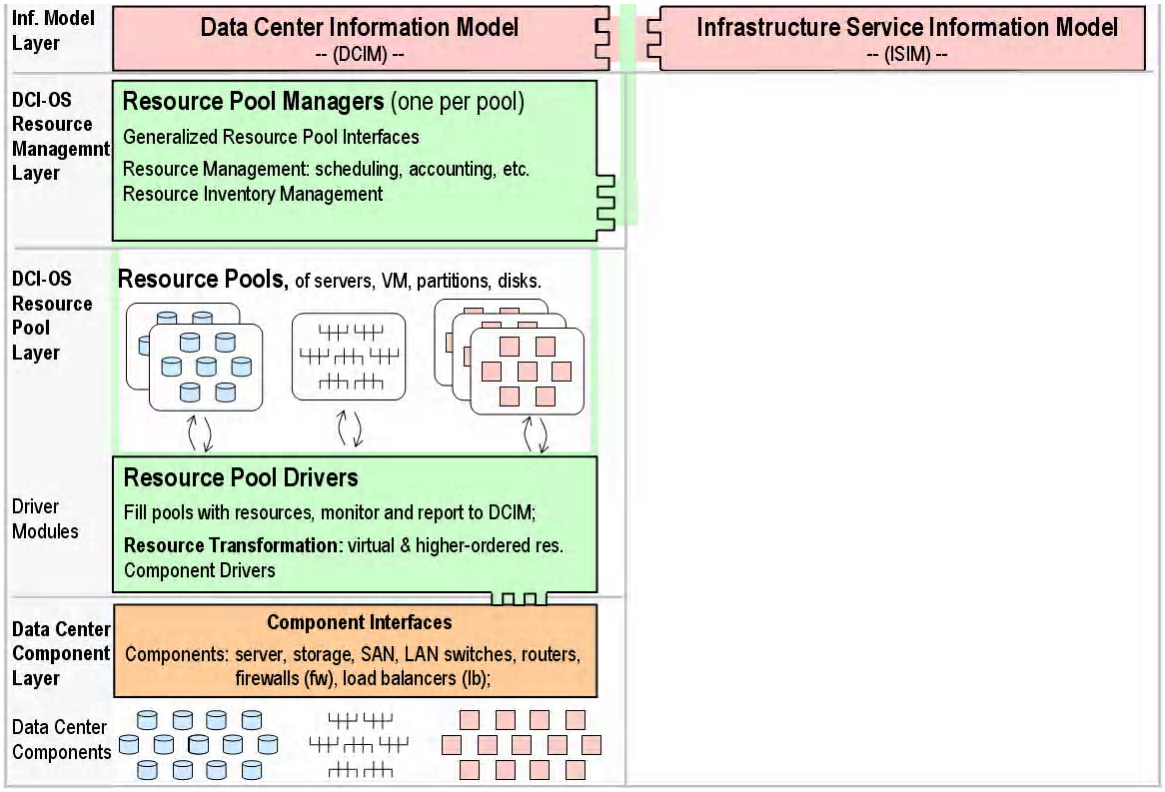


Figure 55: The DCI-OS Layer.

The DCI-OS layer consists of four main sub-layers:

- The *Information Model Layer* with the Data Center Information Model (DCIM) representing the information maintained in the DCI-OS about the data center in which it exists and the Infrastructure Service Information Model (ISIM) representing the information maintained about an Infrastructure Service that exists

in the data center. The Information Model Layer forms the key part of the DCI-OS maintaining the information upon which the modules of the DCI-OS operate.

- The *Resource Management Layer* consists of sub layers of Resource Pool Managers, Resource Pools and Resource Pool Drivers which manage the resources of a data center organized as pools from which resources are allocated and assigned to Infrastructure Services. Pools maintain not only current resource inventory und use. Resource Pool Managers also maintain time profiles to manage the information about past and future resource availability and use.
- Resource Pool Drivers produce the resources for pools, which can be whole or shares of physical resources or can be generated resources, such as virtual resources, which require explicit creation steps.

The *Data Center Component Layer* comprises the physical resources in a data center with management interfaces accessed by Resource Pool Drivers.

This chapter will discuss the information models used in the DCI-OS in detail, specifically in section 9.1 (Information Models in IT Management) – providing a general overview of models used in IT management, section 9.2 (The Common Information Model (CIM)) – providing the overview of the particular framework that was chosen for the DCI-OS, and discuss The DCI-OS Information Model in section 9.3.

Building on the information of these models, a generic Resource Pool Manager is presented in section 9.4 representing the sub-layers of Resource Pool Managers and Resource Pools before section 9.5 concludes the chapter with a discussion of Resource Pool Drivers.

9.1 Information Models in IT Management

The Information Model Layer represents the core part of the DCI-OS since it maintains all persistent information the DCI-OS modules operate upon.

Information can be structured and represented in many different ways. One of the problems in IT management has been that the information maintained in different IT management systems is modeled and represented in very different ways, making integration of management systems hard. One reason for this situation can be seen in insufficient standardization of information models used in IT management. On the one hand, may IT management systems use proprietary data models due to the lack of standards or the lack of development tools implementing standards; on the other hand, there are too many standards that have been proposed by a fragmented community of standards bodies addressing parts of the management domain. Examples will be discussed later. No comprehensive and commonly accepted standard has emerged unifying various data representations and data models in IT management.

As *Model* is understood a set of structured information about elements and their relationships in a management domain that are relevant for management purposes. *Representing* a model refers to storing a model as a structured set of data in a computing system using a syntactic framework (e.g. CIM) where it can be interpreted by algorithms for management purposes. *Interpreting* a model means to reason upon the data set by executing algorithms and deriving actions from its state which can actuate management operations in the management domain.

Models are comprised of:

- Static aspects (elements; e.g. elements in data centers; inventories of hard- and software, licenses, etc.),
- Dynamic aspects (changing properties of elements; e.g. properties of load and utilization conditions, error conditions, etc.).

Model-based Management refers is an architectural pattern used in management systems which separates an environment into two domains:

- the *Managed Domain* – the environment to be managed, and
- the *Management Domain* – the environment of the management.

The Management Domain maintains models about the static and dynamic aspects of the Managed Domain and exercises control over the managed elements by altering control state in models, which is then reflected back as control operations into the managed domain.

Several problems are inherently hard for model-based approaches to IT management:

- How to obtain and create models (by design, by discovery)?
- How to represent models (which framework to chose)?
- How to keep models consistent with the reality they represent?
- How to deal with change?
- How to provide models that not only reflect the current state, but also capture future evolution, anticipation and expectations of future trends, etc.?

Various management frameworks provide various answers to these questions. The following section will discuss some of the information modeling aspects of IT management frameworks.

9.1.1 Frameworks for Information Modeling in IT Management

For the DCI-OS, the use of standards for information models has had a high priority. Standards have mainly emerged and established in particular corners of IT management, such as standards used in network management, e.g. the Simple Network Management Protocol (SNMP) [SNMP] with the Management Information Base (MIB) [MIB] from the OSI/ISO Network management model, or the Common Management Information Protocol (CMIP) [CMIP] which emerged from the ISO/OSI network management model. Industry-specific are the models and protocols defined by the Storage Networking Industry Association (SNIA) [SNIA] for the domain of storage.

Other modeling frameworks have been industry specific, such as TOM [TOM] and eTOM [eTOM] which emerged in the Tele-Management Forum [TMF] and the Open Services Architecture (OSA) [OSA] developed by The Parlay Group [Parlay], which originated for the telecommunications industry.

TMF. The Tele-Management Forum introduced the Telecom Operations Map (TOM) which focuses on the end-to-end automation of communications operations services [TOM]. The core of TOM is a service framework that postulates a set of business processes that are typically necessary for service providers to plan, deploy and operate their services. These processes are organized using the layering concepts of the

Telecommunications Management Network (TMN) [Gal00] and detailed them to a finer granularity. TOM offers concepts and addresses aspects of service management using the abstractions of business processes. The eTOM (enhanced Telecom Operations Map) [eTOM], published by the TMF, is a guidebook, the most widely used and accepted standard for business processes in the telecommunications industry. The eTOM model describes the full scope of business processes required by a service provider and defines key elements and how they interact. eTOM has been adopted by ITU-T as a recommendation and is published in the M.3050.x series.

eTOM is a common companion of ITIL [ITIL], an analogous standard or framework for best practices in information technology.

Both of these frameworks are part of the larger context of Total Quality Management, in which many industries have increasingly formalized their business processes and metrics aiming for better quality, fewer defects, and greater efficiency.

TOM is a reference model for telecommunications networks. It does not define a data model with data structures. The TOM reference model defines the description of core systems and processes involved in the production operation of telecommunications networks. eTOM is tailored as a comprehensive enterprise framework for service providers within the telecommunications industry. eTOM describes all the enterprise processes required by a service provider and analyzes them to different levels of detail according to their significance and priority for the business. For such companies, it serves as the blueprint for process direction and provides a neutral reference point for internal process reengineering needs, partnerships, alliances, and general working agreements with other providers. For suppliers, eTOM outlines potential boundaries of software components to align with the customers' needs and highlights the required functions, inputs, and outputs that must be supported by products.

The eTOM Business Process Framework (BPF) [BPF] encompasses the whole of a telecommunication service provider's enterprise environment. The Business Process Framework starts at the enterprise level and defines business processes in a series of refinements. The framework is defined generically such that it can span organization, technology and services independently and supporting a global community. At the conceptual level, eTOM covers three major process areas:

- Strategy, Infrastructure & Product covering planning and lifecycle management,
- Operations covering the core of operational management, and
- Enterprise Management covering corporate or business support management.

Parlay / OSA. The Parlay Group [Parlay] is an open, multi-vendor consortium formed to develop open technology-independent application programming interfaces enabling Internet and eBusiness services, independently of software vendors, Internet Service Providers (ISVs), software developers, network device vendors, and application service providers to develop applications and technology solutions that operate across multiple networking platform environments.

The Parlay group was formed in 1998. The Parlay group was initiated by a group of operators, IT vendors, network equipment providers, and application developers to support and enable interoperable mobile applications. Parlay defined a number of API specifications, which included limited data models for the information that needed to be

exchanged between mobile devices and service providers. These specifications are standardized with participation of the Parlay Joint Working Group (JWG), which includes the Third Generation Partnership Programs 1 and 2 (3GPP), and the European Telecommunications Standards Institute (ETSI) Services Protocols and Advanced Networks.

Parlay integrates telecom network capabilities with IT applications via secure, measured and billable APIs helping developers avoid rewriting those modules and creating a homogeneous, standardized environment for developing, delivering, measuring, and billing mobile web services.

Web-Services based Management Frameworks. With the advent of web services technologies in the Internet, their advantages in using unified protocols, data representations and interface definitions supporting compatibility and interoperability between applications were recognized for IT management as well. A number of web-services-based standards emerged that were dedicated to enable and support compatibility and interoperability between management applications.

Early representatives emerged as part of the Open Grid Services Infrastructure (OGSI) [OGSI] initiative around 2002 recognizing the fact that open Grids required open management capabilities as well as interoperability between various management systems used in different Grid hosting sites. The web-service based standards originating from this open Grid then merged into the Enterprise Grid Alliance (EGA) [EGA], which was an attempt to evolve those management protocols for use in the enterprise.

After a number of iterations, the efforts finally merged into the Organization for the Advancement of Structured Information Standards (OASIS) standardization body [OASIS] which defined the Web Services Distributed Management (WSDM) [WS-DM] as framework for IT management tasks.

A comprehensive discussion of this topic can be found in *"Web Services in the Enterprise – Concepts, Standards and Management"* [Gra04b].

9.1.2 Behavioral Information Models

While in many cases, information models focus on information about entities and their relationships, behavioral models are an approach to also capture also activity.

Three main categories of behavioral models exist:

- Process models,
- Policy models, and
- Declarative Models.

Process models describe control flows in terms of action flows using the known constructs of sequential and parallel flows, alternatives, cycles and exceptions. A number of process modeling frameworks exists such as the Business Process Execution Language (BPEL) [Ley01], which is also used in data center automation. An overview of processes, process description languages and process management can be found in [Wes07].

Policy models do not describe control flows. Policy models describe conditions and, in case the condition occurs, actions to be taken. Policy-based frameworks have become popular in IT management based on early work in the Event-Condition-Action paradigm [Slo93].

In a policy framework, a policy rule aggregates a set of policy conditions and an ordered set of policy actions. The semantics of a policy rule are such that if the set of conditions evaluates to true, the set of actions is executed.

Policy conditions and actions have two principal components: operands and operators. Operands can be constants or variables. Operators can express both relationships (greater than, member of a set, etc.) and assignments. Together, operators and operands can express a variety of conditions and actions, such as: "**IF** Bob is an Engineer **AND** If the source IP address is in the Marketing Subnet **THEN** Set Joe's IP address to 192.0.2.100 **AND** Limit the bandwidth to 10 Mb".

This represents the classic view to policy in IT management. A number of languages have been defined around this concept. A prominent example is Ponder [Slo01].

Other policy-based frameworks include Policy-Based Network Management (PBNM) [Stra03], Policy-based Admission Control [RFC2753] or IETF's Policy Framework [Hal04]. IETF's Policy Framework working group tried to define a framework within which a wide range of policies can be described and modeled, mainly motivated by a need to represent, manage, share, and reuse policies and policy information in a vendor-independent, interoperable, and scalable manner. The working group has been discontinued in 2004.

IETF RFC 3060 [RFC3060] combines an object-oriented core information model with a representation for policy information. It was jointly developed in the IETF Policy Framework working group and the policy working group in the Common Information Model activity in the Distributed Management Task Force.

The data model defines two hierarchies of object classes:

- structural classes representing policy information and control of policies, and
- association classes that indicate how instances of the structural classes are related to each other.

Association classes can be attributed with policy statements which can be interpreted by a policy engine. The information model can be translated to various concrete implementations, for example, to a directory that uses LDAP as its access protocol.

The policy classes and associations defined in this model are sufficiently generic. Their initial application has been to represent policies related to Quality of Service (QoS) and security standards in the Internet (IPSec) expressed as security policies.

In context of IT management, expression of policy can be classified by their purpose. This classification is useful in querying or grouping policy rules. It indicates whether the policy is used to motivate when or how an action occurs, or to characterize services.

IETF Policy includes following classification of policies:

- *Motivational Policies* are targeted at whether or how a policy's goal is accomplished. Configuration and Usage Policies are specific kinds of Motivational Policies. An example is the scheduling of file backup based on disk write activity from 8am to 3pm.
- *Configuration Policies* define the default setup of a managed entity (for example, a network service). An example of a configuration policy is a set of configuration parameters for an email forwarding service.

- *Installation Policies* define what can and cannot be installed in a system or component, as well as the configuration of the mechanisms that perform the install. Installation policies typically represent specific administrative permissions and can also represent dependencies between different components (e.g., to complete the installation of component A, components B and C are required.
- *Error and Event Policies.* For example, if a device fails between 8am and 9pm, call the system administrator, otherwise call the Help Desk.
- *Usage Policies* control the selection and configuration of entities based on specific usage data. Configuration Policies can be modified or simply re-applied by Usage Policies. Examples of Usage Policies include upgrading network forwarding services after a user is verified to be a member of a certain service group.
- *Security Policies* deal with authentication and permitting or denying access to resources. They include selecting and applying authentication mechanisms and allow accounting and auditing of resources.
- *Service Policies* characterize network and other services. For example, all wide-area backbone interfaces should use a specific type of queuing. Service policies describe services available in the network. Usage policies describe the particular binding of a client of the network to services available in the network.

The following diagram provides an overview of the five central classes comprising the Policy Core Information Model in RFC 3060 [RFC3060], their associations to each other, and their associations to other classes.

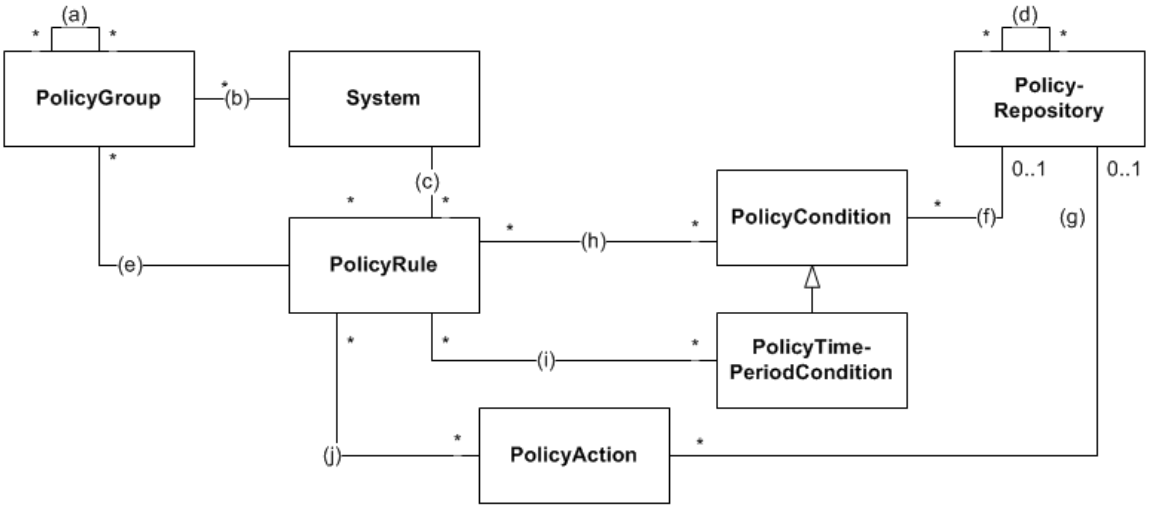


Figure 56: Model of Core Policy IETF Policy [RFC3060].

Figure 56 shows classes related to IETF Policy. The arrows represent the associations:

(a) PolicyGroupInPolicyGroup,

(b) PolicyGroupInSystem,

(c) PolicyRule-InSystem,

(d) PolicyRepositoryInPolicyRepository,

(e) PolicyRuleInPolicyGroup,

(f) PolicyConditionInPolicyRepository,

(g) PolicyActionInPolicyRepository,

(h) Policy-ConditionInPolicyRule,

(i) PolicyRuleValidityPeriod, and

(j) PolicyActionInPolicyRule.

RFC 3703 [RPC3703] defines a mapping of the Policy Core Information Model to a form that can be implemented in a directory that uses Lightweight Directory Access Protocol (LDAP) [LDAP] as its access protocol. This model defines two hierarchies of object classes: structural classes representing information for representing and controlling policy data as specified in RFC 3060, and relationship classes that indicate how instances of the structural classes are related to each other. Classes are also added to the LDAP schema to improve the performance of a client's interactions with an LDAP server when the client is retrieving large amounts of policy-related information.

Declarative models go one step further than policy-based models. They allow the expression of a desired state based on which a controller system determines action sequences leading to a balance with information representing the observed state.

The model developed for the Task Automation Controller presented in section 8.1 is based on declarations of Desired State Model and deriving actions from differences to an associated Observed State Model also fall into this category of policy models.

It is the most advanced approach to a fully automated IT management system which operates based on the pattern of a controller.

9.1.3 Unified Information Model and Information Providers and Consumers

As different management frameworks use different information models, the question emerges how these different information models can be unified, e.g. when management systems must be integrated. There are two fundamental approaches:

- Point-to-point integrations, which require $n*(n-1)/2$ point integrations between n systems or modules, or
- Choosing one information model as a unifying base reducing integrations between n systems to n.

We choose the latter approach for the DCI-OS, and we choose a particular standard as framework for the DCI-OS because of its generality, long-standing history and, most importantly, widespread adoption among IT management vendors: the Common Information Model (CIM) [CIM] which has been standardized by the Distributed Management Task Force (DMTF) [DMTF].

All information referred to in the DCI-OS is folded into a CIM representation. System participating in a DCI-OS, e.g. a monitoring system, must render their data into a CIM representation. In many cases, this means to build information wrappers in order to turn any data source or sink into a CIM information provider.

Central part of the DCI-OS Information Model Layer is the CIM repository into which a number of modules provide information, and from which modules of the DCI-OS also consume information. Any module of the DCI-OS, which includes external management

systems, acts as a CIM provider to the CIM repository. Section 8.2 has demonstrated this principle for the integration of the (existing, proprietary) Radia system into the DCI-OS as the Deployment Manager.

Models or modeled information can be supplied by modules as information providers, which is typically the case for dynamic information such as states and conditions or discovered information such as inventories. Models or modeled information can also be provided manually by people, which is mainly the case for structural model definitions (model schemata) or modeled information about past or future states, such as future designs of a Resource Topology for an Infrastructure Service. Tools can assist people create these models.

In order to structure the large area of information that is relevant in IT management, a sole syntactic modeling framework is not sufficient. What is also needed is a methodology which allows representing and structuring the various information types in a clear manner. CIM refers to the object-oriented methodology for its definitions.

The Common Information Model provides such a methodology for system-related information entities (called *Managed Element*) using an object-oriented methodology.

The methodology used in CIM allows that functional, organizational and life cycle aspects to be considered are addressed by:

- Generic and abstract service definition: The model gives an abstract definition of a service and thus provides a common understanding for describing services dependencies for a particular scenario or environment.
- Integration of organizational aspects: The modeling approach defines a service as the association between organizations that provide and use services. It allows to model scenarios such as supply chain and provider hierarchies.
- Separation of service definition and service implementation: The separation of the abstract service description from the corresponding service implementation enables providers to implement services according to their local environment without imposing changes on client services (service virtualization).
- Management as an integral part of the service: the model considers the management of services as an integral part of the service itself.

Over the years, a substantial hierarchy of entity classes has been developed for CIM (with currently over 3,500 entity classes for the same number of component types that are managed by IT systems). The object-oriented hierarchy allows structuring the various entity types at different levels of abstraction.

Models are bound to a domain. In case of CIM, this domain reflects the technical aspects of a managed environment. The top-level concept is a *Managed Element*. Organizational aspects, such as people in responsible roles, are not directly addressed in CIM's core model since they are not managed elements. Furthermore, complex information dependencies may span beyond the technical aspects, e.g. information about people in responsible roles for certain management systems, organizations who are in charge of handling incidents, etc. Information can become complex very quickly crossing multiple organizational, administrative and jurisdictional boundaries leading to inter- and intra-organizational dependencies.

Since the top layer of the DCI-OS is a Planning and Design Layer, which includes people, organizations, plans and designs, CIM had to be extended by a number of concepts which is discussed in section 9.3 (The DCI-OS Information Model) in more detail.

9.2 The Common Information Model (CIM)

The Distributed Management Task Force (DMTF) defines and develops the Common Information Model (CIM) as an ongoing activity. CIM introduces a management information model that allows integrating the information models of existing management architectures making them act as information providers. CIM's purpose is to act as a unifying information base that allows exchanging management information between arbitrary management systems.

The Common Information Model has established an information model that spans across domains, including compute servers, storage and networks providing CIM a unique advantage over other frameworks that are domain specific. CIM provides consistent definitions and structures for data. The language that CIM uses for external representation is the *Managed Object Format (MOF)*. CIM is independent of specific implementations and is not tied to any particular database or information repository.

The DMTF Common Information Model (CIM) is an approach to the management of systems and networks that applies the basic structuring and conceptualization techniques of the object-oriented paradigm. The approach uses a uniform modeling formalism that—together with the basic repertoire of object-oriented constructs and supports the cooperative development of an object-oriented schema across multiple organizations.

The management schema of CIM is divided into three conceptual layers:

- *Core model* – an information model that captures notions that are applicable to all areas of management.
- *Common model* – an information model that captures notions that are common to particular management areas, but independent of a particular technology or implementation. The common areas are systems, applications, databases, networks and devices. The information model is specific enough to provide a basis for the development of management applications. This model provides a set of base classes for extension into the area of technology-specific schemas. The Core and Common models together are expressed as the CIM schema.
- *Extension schemas* – represent technology-specific extensions of the Common model. These schemas are specific to environments, such as operating systems (for example, UNIX or Microsoft Windows).

The **Core Model** gives a formal definition of a service and allows hierarchical and modular composition of services consisting of other services. However, the focus in CIM is on rather technical details of components than of service composition and does not include a notion of domains, such as customer and provider.

The **Common Model** is a conceptual model for describing a business computing and networking environments with all the managed entities, their states, operations, composition, configuration, and relationships. Model contents are not bound to a particular problem domain or implementation, but address end-to-end management from clients, to servers, and over the network.

The **Extension Schemas** allow to extend the Common Model in case information is needed which cannot be represented in the Core or Common Models.

CIM reflects the principles of FCAPS management (fault, configuration, accounting, performance and security management) and supports the abstraction and decomposition of services and functionality. The information model defines and organizes common and consistent semantics for managed entities.

The organization of CIM is based on an object-oriented paradigm promoting the use of inheritance, relationships, abstraction, and encapsulation to improve the quality and consistency of management data. Object-orientation in CIM is applied in the following dimensions:

- *Abstraction and classification.* To reduce the complexity, high level and fundamental concepts (the objects of the management domain) are defined. These objects are grouped into types of management data (classes) by identifying their common characteristics and capabilities (properties), relationships (associations) and behavior (methods).
- *Object inheritance.* Additional detail can be provided by sub-classing. A subclass inherits all the information (properties, methods and associations) defined for its super class. Subclasses are created to classify the levels of detail and complexity at the right level in the model.
- *Dependencies, component and connection associations.* Being able to express relationships between objects is an extremely powerful concept. Before CIM, management standards captured relationships in form of multi-dimensional arrays or cross-referenced data tables. Relationships and associations are now directly modeled. In addition, the way that these relationships are named and defined, gives an indication about the semantics of object associations. Further semantics and information can be provided in properties.

9.2.1 CIM Core and Common Model

The core model comprises of a basic set of classes, properties and methods that provide the basis for modeling of all managed systems. The Core and common models follows the CIM Meta-Model. The Core and Common Models are together referred to as CIM Schema. The CIM Common Models capture information related to particular areas of management like systems, applications, networks, devices etc.

The core model lays down a basic classification of elements and associations. The managed element class which is an abstract class forms the base of the hierarchy. The Managed Element class is sub-classed into Managed System Element, Product related, Setting and Configuration related performance and statistical data related classes. Managed system elements are sub-classed further into Logical and Physical Elements.

The common models capture the information that is related to a management area but is technology and platform independent.

The CIM Schema [CIMv2.7] defines the schemata for the Common Model based on the following sub-classification of the central concept of *Managed Element*:

1. *Applications:* Deals with the structure of an application, life cycle and the transition between states in the life cycle of an application.

2. *Database:* This schema describes the database system that describes the application software, the logical entity of the database and the database service.
3. *Devices:* The CIM Device Common Model describes a range of hardware, their configuration information and their data. It covers concepts like sensors, fans and batteries to storage volumes.
4. *Events:* The CIM Event Common Model covers the aspects of publications, subscriptions and notifications.
5. *Interop:* The CIM InterOp Model describes the various Web Based Enterprise Management (WBEM) components, namely the CIM Client, CIM Server, CIM Object Manager, Provider. Please see more details in the next subsection.
6. *Metrics:* The CIM Metric Common Model tries to generalize the concept of transactions through the UnitOfWork concept.
7. *Network:* The CIM Network Model describes the network systems, network services, logical interconnections and accesses, network protocols (OSPF, BGP), networking technologies (VLAN, Switching/Bridging), Quality of Service (meters, markers, queues) and other related definitions.
8. *Physical:* The CIM physical Common Models describes the modeling details of Physical elements (elements that occupy space and follow laws of physics). CIM_Racks, and CIM_Chassis are defined for example as physical elements.
9. *Policy:* Policies are frequently describes as rules that change the behavior of a system. The Policy Model has been developed jointly by IETF and DMTF. It expresses policies as condition action pairs. The <condition> term is a Boolean expression used to specify the rule selection criteria. When for a resource the <condition> term evaluates to true the <action> term is invoked.
10. *Support:* The CIM Support model deals with standardized way to represent and communicate information, the process of obtaining information, publishing and interpreting support information.
11. *Systems:* CIM System Common Model defines computer system related abstractions. These systems are aggregation entities and are not modeled as collections. They also deal with concepts like systems, files, operating systems, processes, jobs, etc.
12. *User:* The CIM User Common Models deal with the general contact information related to users, organizations, etc. and the clients of the services as "Users" and the security authentication and authorization related information.

9.2.2 CIM Meta-Model

The CIM Schema describes the core and common models. CIM is an object oriented model and has capabilities to represent types, and instances alike. CIM defines the notion of Classes and sub-Classes. Classes are types while subclasses are subtypes. Instances are instantiation of the Classes and subclasses and represent things. Properties are attributes and Relationship is pairs of attributes.

The elements of the model are Schemas, Classes, Properties and Methods. The Unified Modeling Language (UML) [Boo98] is used to define the structure of the meta-schema.

The model also supports Indications and Associations as types of Classes and References as types of Properties.

- A *Schema* is a group of classes with a single owner. Schemas are used for administration and class naming. Class names must be unique within their schema.
- A *Class* is a collection of instances that support the same type: that is, the same properties and methods.
- Classes can be arranged in a generalization hierarchy that represents subtype relationships between Classes. The generalization hierarchy is a rooted, directed graph and does not support multiple inheritance.
- Classes can have Methods, which represent the behavior relevant for that Class. A Class may participate in Associations by being the target of one of the References owned by the Association. Classes also have instances (not shown in the figure).
- A *Property* is a value used to characterize instances of a Class. A Property can be thought of as a pair of Get and Set functions that, when applied to an object, return state and set state, respectively.
- A *Method* is a declaration of a signature (that is, the method name, return type and parameters), and, in the case of a concrete Class, may imply an implementation.
- A *Trigger* is a recognition of a state change (such as create, delete, update, or access) of a Class instance, and update or access of a Property.
- An *Indication* is an object created as a result of a Trigger. Because Indications are subtypes of Class, they can have Properties and Methods, and be arranged in a type hierarchy.

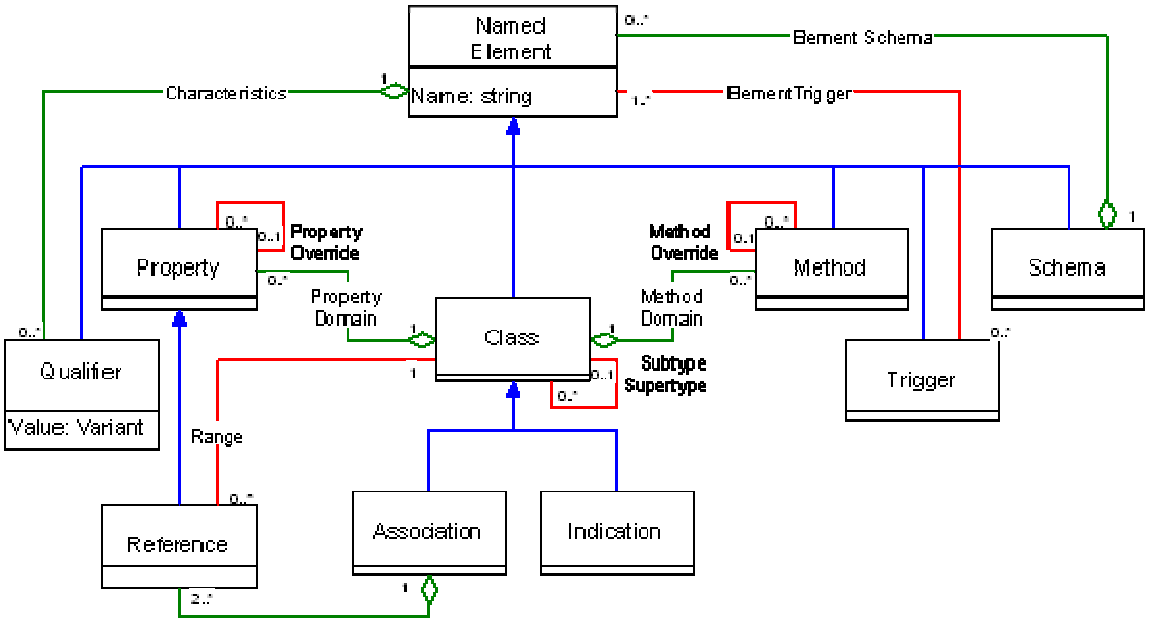


Figure 57: UML diagram of the CIM Meta-model.

- An *Association* is a class that contains two or more References. It represents a relationship between two or more objects. Because of the way Associations are

defined, it is possible to establish a relationship between Classes without affecting any of the related Classes. That is, addition of an Association does not affect the interface of the related Classes. Associations have no other significance. Only Associations can have References. An Association cannot be a subclass of a non-association Class. Any subclass of an Association is an Association.

- *References* define the role each object plays in an Association. The Reference represents the role name of a Class in the context of an Association. Associations support the provision of multiple relationship instances for a given object. For example, a system can be related to many system components.
- *Properties and Methods* have reflexive associations that represent Property and Method overriding. A Method can override an inherited Method, which implies that any access to the inherited Method will result in the invocation of the implementation of the overriding Method. A similar interpretation implies the overriding of Properties.
- *Qualifiers* are used to characterize Named Elements (for example, there are Qualifiers that define the characteristics of a Property or the key of a Class). Qualifiers provide a mechanism that makes the meta-schema extensible in a limited and controlled fashion. It is possible to add new types of Qualifiers by the introduction of a new Qualifier name, thereby providing new types of meta-data to processes that manage and manipulate classes, properties and other elements of the meta-model. See below for details on the qualifiers provided.

A CIM Class is a blueprint that describes the properties and methods of a particular type of an object. Classes have properties and methods. A Class Name is unique in a particular schema. Properties are unique within a class. A property has a name, a data type, a value and an optional default value. The data types that CIM supports are uint8, sint8, uint16, sint16, uint32, sint32, uint64, sint64, string, boolean, real32, real64, datetime, ref (reference to a class name), char16.

Methods are unique within a class and represent operations that can be invoked. Return type of methods conform to data types supported by CIM. A method signature involves name of the method, return type, optional input parameters, and optional output parameters. Return types must not be arrays.

Qualifiers are used to describe additional information about classes, associations, properties, methods, indications, properties or references. Qualifiers have name, type, flavor, scope, and optional default value.

An Association has references to two or more classes. Associations represent relationships between two or more classes. It is treated as a separate object with references attached to it.

An Indication signifies occurrence of an actual event. They may have properties and methods and be hierarchically arranged. The indications are either life cycle indications or process indications. The life cycle indications signify creation, deletion or modification of a class or a creation, deletion, modification, method invocation, and read access of an instance. The process indications are other notifications that are not related to life cycle. A CIM Specification is described in the Managed Object Format (MOF).

9.2.3 Managed Object Format (MOF)

The management information is described in a language based on Interface Definition Language called the Managed Object Format (MOF). This document uses the term MOF specification to refer to a collection of management information described in a manner conformant to the MOF syntax. Elements of MOF syntax are introduced on a case-by-case basis with examples.

The MOF syntax is a way to describe object definitions in textual form. It establishes the syntax for writing definitions. The main components of a MOF specification are textual descriptions of classes, associations, properties, references, methods and instance declarations and their associated qualifiers. Comments are permitted.

In addition to serving the need for specifying the managed objects, a MOF specification can be processed using a compiler. To assist the process of compilation, a MOF specification consists of a series of compiler directives.

```
//==========================================================/
// A class example: ManagedElement
//==========================================================/
[Abstract, Version ("2.7.0"), Description (
   "ManagedElement is an abstract class that provides a common "
   "superclass (or top of the inheritance tree) for the "
   "non-association classes in the CIM Schema.") ]
class CIM_ManagedElement {
   [MaxLen (64), Description (
      "The Caption property is a short textual description (one-"
      "line string) of the object.") ]
   string Caption;
      [Description (
      "The Description property provides a textual description of "
      "the object.") ]
   string Description;
       [Description (
      "  A user-friendly name for the object. This property allows "
      "each instance to define a user-friendly name IN ADDITION TO its "
      "key properties/identity data, and description information. \n"
      "  Note that ManagedSystemElement's Name property is also defined "
      "as a user-friendly name. But, it is often subclassed to be a "
      "Key. It is not reasonable that the same property can convey "
      "both identity and a user friendly name, without inconsistencies. "
      "Where Name exists and is not a Key (such as for instances of "
      "LogicalDevice), the same information MAY be present in both "
      "the Name and ElementName properties.") ]
   string ElementName;
};
```

Figure 58: Example of a MOF description in the Common Information Model.

9.3 The DCI-OS Information Model

This chapter introduces the core information model for the DCI-OS. It is based on the models described by the author in the *Conceptual Architecture*, which forms the basis for the DCI-OS Information Model [Gra03b].

Concepts establish the terms (the vocabulary), abstractions and principles for the DCI-OS Information Model. Concepts reflect the basic abstractions upon which the DCI-OS modules operate. Eight such basic abstractions are defined in this chapter. They are referred to as first-class entities.

The set of eight first-class entities is meant to be *complete*. All information needed to be maintained by a data center operating system can be classified into these base abstractions. All functions performed by such an operating system can be mapped into operations about instances of first-class entities. First-class entities are *orthogonal*.

First-class entities also provide the basis for the models used by the DCI-OS. They define the complete set of abstraction upon which the DCI-OS modules perform their operations. Some of the first-class entities are well known from other modeling environments such as CIM. Examples are resources, relationships, policy and activities. Others are actors and roles in which actors operate in the system. Two concepts have been introduced providing structural means. These are contexts and views. Those concepts are not typically defined in modeling environments. In the Information Model, they are first-class entities with explicit representation in the system.

The following sections define the set of first-class entities as well as their relationships to each other.

9.3.1 First-Class Entities

A system of first-class entities defines a canonical set of entity types that are relevant for a purpose and thus need to be represented in a system for reasoning about them. Canonical refers to a minimal and complete set of types that are semantically orthogonal.

Entity – An entity is a thing or a concept that is relevant for the operation of a system and thus needs representation inside the system.

An entity representation is the implementation of the entity information inside the DCI-OS system in a chosen data representation model. An entity type describes a category of entities of the same kind.

Figure 59 shows eight concepts as first-class entities and sub-classes of class entity.

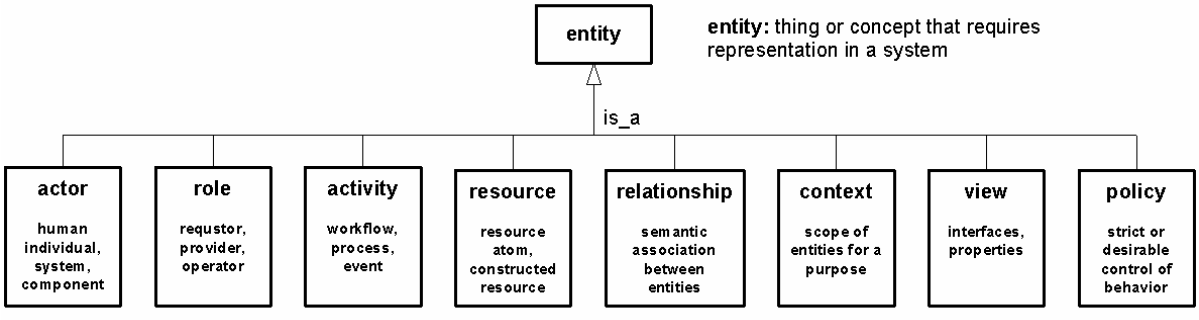


Figure 59: First-class entities of the DCI-OS Information Model.

Entity Type	Purpose
Actor	subject in the DCI-OS initiating or performing activity, including (human) individuals, systems or components.
Role	defines the purpose of an actor in a system which determines the views that actor has on other entities.
Activity	flow of operations or actions associated with a task that is initiated by one actor in one role.
Resource	object in the DCI-OS: resource atom or constructed resource.
Relationship	semantic association between entities.
Context	structural element that defines a scope within which semantically related entities exist, have meaning and interact to accomplish a purpose.
View	appearance of entities to roles in terms of properties and operations.
Policy	externally imposed strict or desirable control over behavior of actors or activities when operating upon entities.

Table 9: First-class entities as conceptual basis of the DCI-OS Information Model.

Table 9 summarizes the first-class entities and categorizes them with their relationships to associated entities in the managed and/or management environment.

Entities provide the basic building blocks for structuring the information about things and concepts needed for the operation of a DCI-OS system. Reasoning upon entities controls the behavior of the DCI-OS with all internal and external interactions.

9.3.2 Relationships Between First-Class Entities

Figure 59 showed the set of first-class entities as subclasses of entity. Other relationships among first-class entities are shown in the following figures.

In Figure 60, the first-class entities of the DCI-OS Information Model are shown. The DCI-OS is represented as an actor with which other actors interact such as by requesting resources.

Actors are subjects in a DCI-OS that initiate or perform activity such as requesting a resource or maintaining a resource inventory. Actors initiate activities in roles. An actor may have multiple roles or may change roles for different activities. One activity is initiated with one role. Human individuals, processes, components, entire systems are included in the notion of an actor. An allocation system or an assignment system may act as actors in the DCI-OS. Resources can be active initiating and performing operations upon themselves. Examples of such operations are requesting more resources or releasing resources.

Roles define the purpose of an actor in a system which determines the views (properties and operations) that actor has on other entities. Roles are defined by contexts in which

actors act within or interact with the DCI-OS. Roles define the views actors and activities have on other entities in the system. Examples of roles are requestors, providers or operators of resources in a DCI-OS.

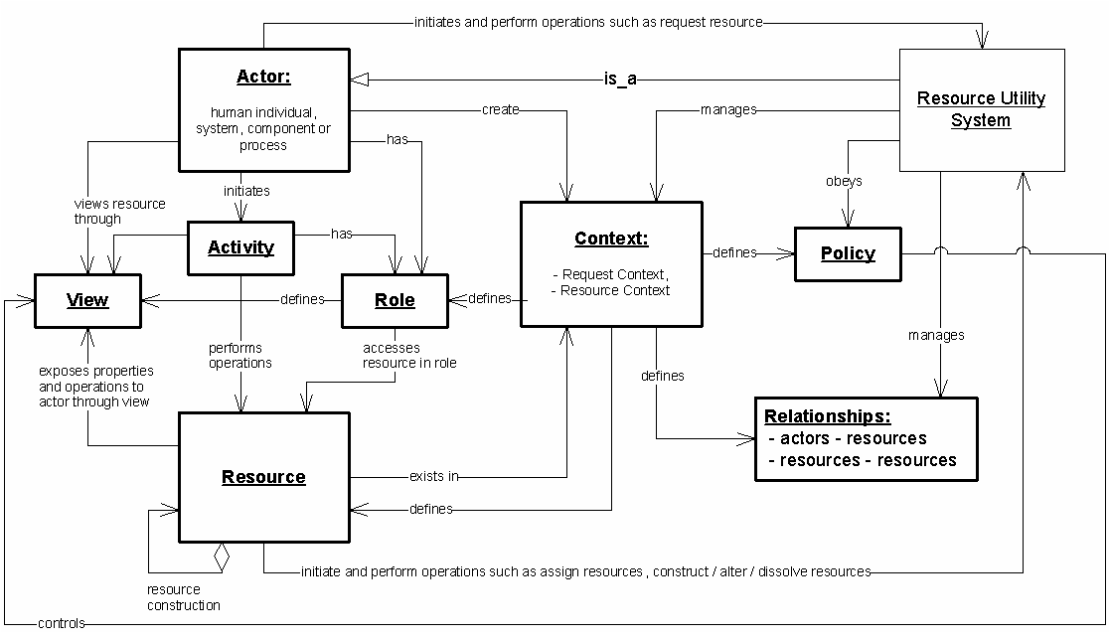


Figure 60: Top-level relationships among first-class entities.

Activities encompass the flows of operations or actions associated with tasks that are initiated by actors in the DCI-OS. Workflows, processes or events are examples of activities.

Resources are the main objects in a DCI-OS that are provided, controlled and used by actors for performing computational services. Resources can a priori exist in resource pools as resource atoms, or resource can be constructed creating new resources with higher abstractions. Resource construction may define new contexts within which new resources appear that can be provided, controlled and used by actors.

Relationships define semantic associations between internal or external entities in the DCI-OS. Relationships are managed by the DCI-OS, primarily relationships between actors and resources (such as resource allocation and resource assignment relationships) and between resources (resource constructions as resources that are composed of other resources). The entire operation of a DCI-OS can be seen as the creation, transitioning and dissolving of entities and managing relationships among them.

Contexts, as structural elements defining scopes for semantically related entities, are used to organize all information maintained in the DCI-OS about entities. The DCI-OS manages context entities and entities within contexts. Contexts characterize the domains within which entities exist and have meaning. Scoping may also define visibility bounds of entities. Resource contexts define the relationships among resources for resource construction. Contexts may also define policy applying to entities in those contexts. An example of a context is a resource request context, which contains all information

associated with one resource request from one requestor. Another example is a resource context containing all information associated with one (eventually constructed) resource. Actors create, transition and dissolve contexts in the DCI-OS.

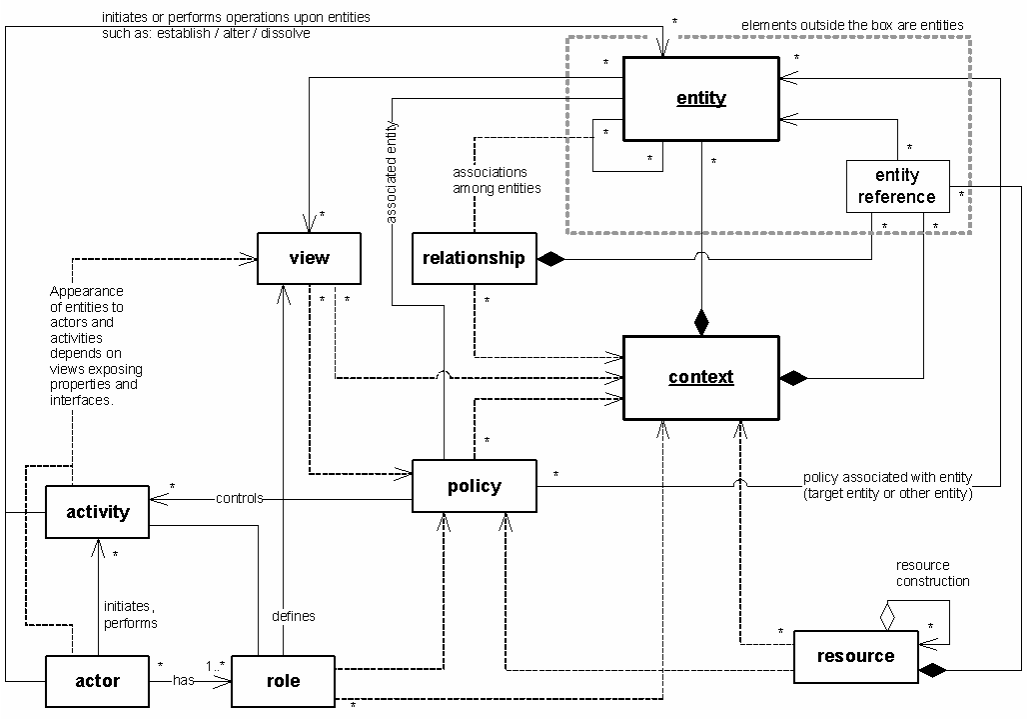


Figure 61: Refinement of relationships among first-class entities.

Views define the appearance of associated entities to roles by exposing certain properties and operations to those roles in which actors and activities interact with entities.

Policy defines strict or desirable control over behavior of actors and activities when performing operations upon entities. Policy is associated with entities to which it applies. Actors and activities inside the DCI-OS obey policy.

Computations upon entity representations drive the operation of a DCI-OS. New entity representations are created, for instance, when new resources are created or new users are registered. Resource allocations and assignments are represented as relationships between requestor and resource entities inside the DCI-OS. Complex transactions may occur when new resources are constructed or when resources are assigned to requestors.

Entity representations are created within contexts by instantiation from entity types. Entity types have themselves representation as entities in the system.

The entirety of entity representations summarizes all information the DCI-OS maintains about all entities that are associated with its operation. The DCI-OS maintains consistency of information stored in entity representations.

The following sections provide detailed definitions of first-class entities. Definitions are provided first for entities as things or concepts and second for corresponding entity representations as the information maintained inside the DCI-OS about entities.

9.3.3 Entity Type: Actor

Actor – An actor is an acting subject that initiates or performs activity in the system. Human individuals as well as machine-based actors such as systems, components or processes are included in the notion of an actor. Actors are associated with roles defined by the context within which they operate. Actors may have different roles performing different activities in the system in different contexts.

Actor Entity – An actor entity contains the information that is represented about an actor in the system.

Actors initiate and perform activity in the system. Different kinds of actors may perform different kinds of activities.

Examples of Actors	Examples of Activity
Human operator	Manage resource pool.
Application process	Request resources.
Resource request workflow	Route resource request through the DCI-OS.
Resource allocation system	Perform resource allocation algorithm, allocate resources to requestor.
DCI-OS itself	Allocate, assign, and manage resources.
Event detector	Send notification to management system.
Management system	Collect monitoring data.

Table 10: Examples of actors and related activities.

9.3.4 Entity Type: Role

Role – A role defines the purpose of an actor in a system. A role determines the views an actor (and initiated activity) has on other entities it is interacting with. Views depending on roles define a set of operations and properties of an entity that is exposed to an actor and its activities. Roles depend on the context within which interaction occurs.

Role Entity – A role entity contains the information that is represented about a role in the system.

In many systems roles are not distinguished from actors ("users"). In Unix, for instance, permissions depend on the user identification. It is not supported that a user under the same user identity can act as regular user or system administrator. In order to perform the role of an administrator, the user identification must be changed.

In a DCI-OS, actors are detached from roles. Actors thus can use the same identification, yet act in different roles when permissions allow. Each actor must have at least one role associated within which activity is performed. Actors may change roles (when permissions allow).

Actors may initiate multiple activities in the system in different roles such as an operator (actor) managing a resource pool (operator role) and at the same time allocating resources (in requestor role) for deploying and operating a management system.

Roles determine the views under which entities appear to actors. Examples of roles are: operator role, resource provider role, or resource requestor role.

9.3.5 Entity Type: Activity

Activity – Activity is a flow of related operations or actions to accomplish a task that is initiated by one actor in one role. Actors may also perform activity. Activity causes state transitions in the system. Activity may be passed between actors.

Activity Entity – An activity entity is the representation of an activity in the system. An activity entity contains information such as identification of the activity, the actor who initiated and who is performing activity, the program that defines the sequence of operations performed by an activity, and the current status of an activity.

Not all activities in the system are represented as activity entities.

Examples of activities are workflows, processes, events, notification chains, in general all sequences of operations that serve the purpose of accomplishing a task.

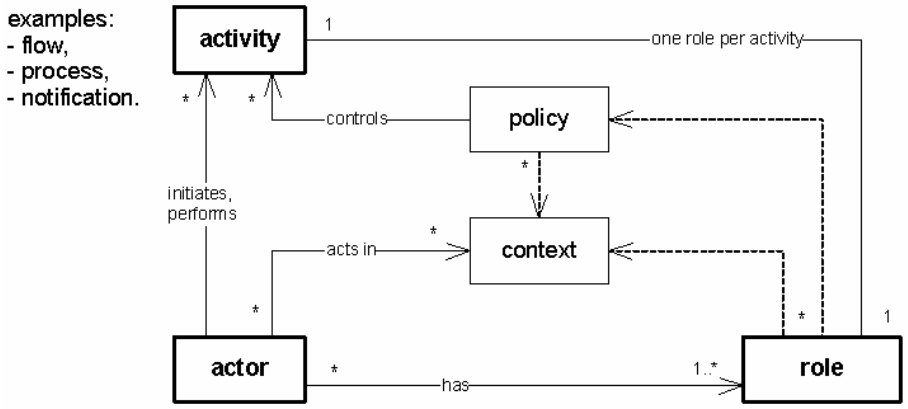


Figure 62: Relationships between actor, activity, and role.

The role defines the view and thus the required privileges under which entities appear. Multiple actors may act in same roles, and one actor can take multiple roles simultaneously or subsequently initiating different activities. One activity is initiated in one role. Contexts define roles. Activity is associated with the role within which it is performed. Actors performing activity may change over time.

Roles are associated with actors either implicitly by acting within a certain context that implicitly selects the role for actors in that context, or a role can be selected explicitly by an actor (when permission policy allows). An example for the first case is a user requesting resources from the DCI-OS. Requesting resources implicitly implies a requestor role.

9.3.6 Entity Type: Relationship

Relationship – A relationship defines a semantic association between entities. The DCI-OS manages creation, transitioning and dissolving of relationships.

Relationship Entity – A relationship entity contains the representation of a semantic association between entities in the system in form of structured sets (pairs, tuples, ordered lists, etc.) of references to entities between associations exist.

Not all relationships are explicitly represented as relationship entities.

The entire operation of a DCI-OS is based on creation, transitioning state and dissolving of entities and creating, transitioning and dissolving relationships among entities. Transitions among entities may be complex requiring transactional semantics.

Relationships can be established between any entity, including relationships. Relationships primarily exist between actors and roles, actors and resources, and among resources establishing constructed resources.

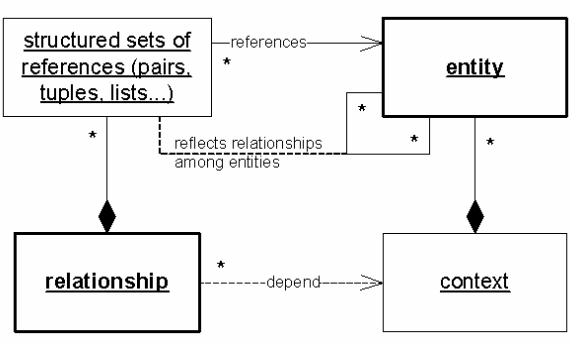


Figure 63: Relationship entity.

For example, resource allocation is defined as the process of establishing allocation relationships between a set of resources and a resource request issued by an actor. Resource construction is the process of establishing relationships among resources. A resource topology is another example of explicit relationships established among resources representing resource wiring and connection information.

A relationship entity exists as an explicit, identifiable, namable entity representation within a context. Relationship entities can be explored from outside. Not all relationships are represented as explicit entities in the system. Some relationships may be implicit.

9.3.7 Entity Type: Context

Context – As context is understood a structural element that defines a scope within which semantically related entities exist.

Context Entity – A context entity provides the container where semantically related entities or references to entities are maintained in the system.

All first-class entities are contained in context entities including recursively sub-contexts. Contexts may also contain references to entities that exist in other context entities. One entity can only exist in one context at a time, but may be referenced multiple times from one or multiple contexts.

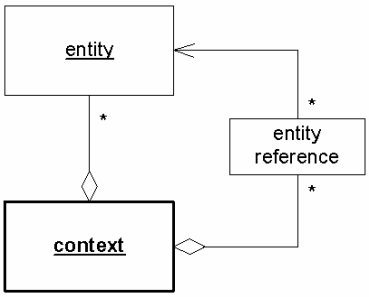


Figure 64: Context entity containing other entities or entity references.

Examples of context entities are:

- *resource request context entity* – contains a set of requested resources along with time lines and other descriptions for the request; later it also contains the resource entities that have been allocated and assigned to the request that are passed back to the requestor for access.
- *resource pool context entity* – contains resource atoms and resource constructions of one particular resource pool.
- *allocation context entity* – contains allocation relationship entities between resource requests (context entities) and resource entities.
- *assignment context entity* – contains assignment relationship entities between resource requests (context entities) and resource entities.

The resource pool context entity primarily contains resource entities maintained in a particular resource pool (atoms or constructions). The purpose of the resource pool context entity is to group and scope these entities to the resource pool and distinguish them from other resource pools. Further entities such as actors and roles (requestors, operators) can be added to the resource pool context.

Entities can be referred to from multiple contexts, and thus become semantically part of other contexts, but they can only exist as entity in one context.

Contexts can be established dynamically. An example is a resource request that is represented as a resource request context entity that initially contains resource descriptions (as form of resource entities) and later references to allocated and assigned resources that exist or are established in resource pool contexts. Requestors are informed about assigned resources be returning their resource request context entity which, at that stage, contains references to the assigned resources. The requestor then uses these references to access resources.

Resource request context entities are the primary entities used for the interaction between requesting actors and the DCI-OS.

Entities may transition from one context entity to another. Following operations exist for entities maintained in contexts:

- creating entities within contexts,
- joining or leaving contexts and transfers between contexts,
- establishing references to entities in contexts,
- establishing view and relationship entities to associated entities.

9.3.8 Entity Type: View

View – A view defines the appearance of associated entities to roles by exposing certain (new, changed) properties and operations to roles in which actors and activities interact with associated entities. A view only establishes a new appearance of associated entities, not duplicates of states.

View Entity – A view entity establishes a new set of transformed properties and interfaces upon associated entities. A view entity contains a set of transformation functions and relationships to the associated entities in the system.

Views are useful to distinguish and control operations actors are allowed to perform with entities depending on their roles. For example, an operator role would view a resource entity differently than a requestor role.

Hiding properties or operations through views is another useful example of transformations (subset) in view entities. Not only smaller, also extended properties or interfaces can be derived from associated entities as result of transformations.

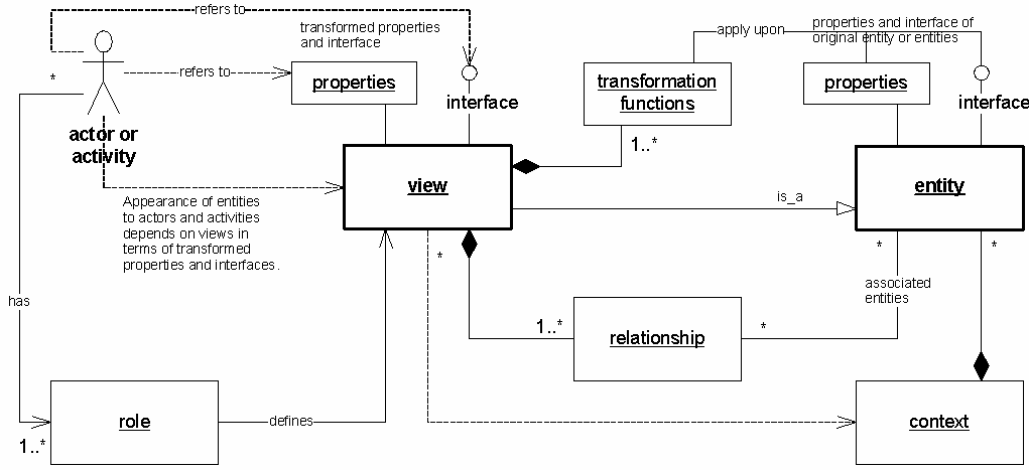


Figure 65: Views as transformations of entity properties and interfaces.

View entities contain transformation functions that are based on original properties and interfaces of one or more associated entities. Views expose new properties and interfaces by applying transformation functions defined by a view entity.

In contrast to resource construction where aggregation or transformation processes established new resource entities, views only establish new appearances in terms of transformed properties and interfaces of existing entities.

Views do not maintain state of associated entities. Accessing state of associated entities has to be performed though view entities where transformation functions then access state in associated entities.

9.3.9 Entity Type: Policy

Policy – As policy is understood the entirety of strict (enforced) constraints or desirable directives that control the behavior of a target entity towards achieving a goal. Target entities are the entities to which policy applies. Policy is imposed on a target entity either externally (provided from outside) or internally (configured or built-in). Policy includes how decisions are made in or about target entities and the actions following from decisions. Policy thus controls the behavior of an actor or an activity operating upon a target entity. Behavior is controlled within a space of choices that is exposed to policy control.

Policy Entity – A policy entity contains the representation of policy in some form in the system such as in form of interpretable specifications of policy or constraint languages, instruction statements, sets of rules, or sets of control variables. Policy representation is interpreted and obeyed by actors and activities operating upon a target entity.

Policy entities are associated with either target entities (for internal policy) or with external entities carrying policy information to be applied when interaction with a target entity occurs.

Examples of internal policy are allocation and assignment policy that applies in a resource pool. An example of external policy is a preference expressed in resource request for prioritized treatment. The internal policy of a DCI-OS decides whether to honor or disregard such external policies.

Policies are represented as policy entities and are attached to the entities for which they apply.

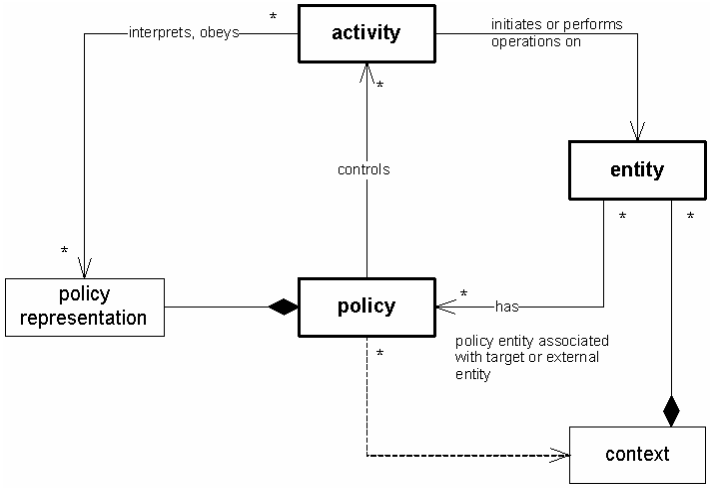


Figure 66: Policy entity with policy representation.

Properties of policy include:

- Policy controls behavior of a target entity in form of strict (enforced) constraints or desirable directives. Policy controls the behavior of a target entity within a space of choices that is exposed to policy control in a target entity.
- Policy is directed towards achieving a goal that has been defined for a target entity and formulated in form of a policy specification.
- Policy can be imposed internally to a target entity in form policy entities associated with the target entity. Examples are built-in policies or policies that have been pre-configured for a target entity.
- Policy can be imposed externally to a target entity when interacting with outside entities providing policy information for interactions with target entities.
- Behavior of target entities is controlled by deriving decisions from policy representations contained in policy entities and the actions following from decisions.
- Policy specifications are interpreted and obeyed by activities when performing operations upon target entities.
- Policy is represented in some form in policy entities and associated with either target entities as internal policy or with external entities and applied when interactions occur with a target entity.

- Examples of policy representations include:
  - statements of policy or constraint languages,
  - statements of code and flow descriptions,
  - sets of rules (such as condition-action pairs),
  - sets of control variables defined for a decision space.

Policy representation may be translated from one representation into another.

9.3.10 Entity Type: Resource

Resources are the objects in the DCI-OS. Resource entity representations maintain construction relationships with other resource entities and maintain a reference to the associated resource(s) that exist(s) in the resource pool. Associations are provided in form of relationship entities contained in resource entities. Resource atoms are resource entities that are associated with resources that a priori exist in a resource pool and have not been subject to construction by the DCI-OS.

Resource – A resource provides a computational service in terms of processing, storage, or communication that is needed by another computation. Resources are the objects in the DCI-OS that are provided, constructed, controlled and used by actors. Resources can be classified into atomic resources and constructed resources. Resource assignment relationships and resource construction relationships are primary relationships managed by a DCI-OS.

Resource Pool – A resource pool is a bounded domain where a set of resources is located that is managed by a DCI-OS. The DCI-OS provides an interface through which requestors can explore, request and use resources from the pool. Multiple resource pools can form a federation managed by the DCI-OS providing unified views and ways to access resources from different pools.

Resource Entity – A resource entity is the representation of a resource in the system. A resource entity provides interfaces for controlling and maintaining status data of an associated resource. A resource entity internally maintains construction relationships and references to associated resources.

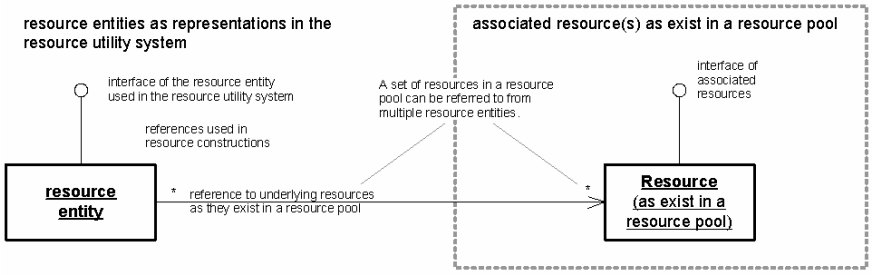


Figure 67: Resource entity and associated resource in a resource pool.

The following table summarizes the classification of resources that has been chosen for a DCI-OS. It contains a brief characterization and examples of resources of each category. More detailed discussion of terms and definitions follows after.

The major distinction shown in Table 3 appears between resources that a priori exist in a resource pool and resources that are results of constructions upon other resources.

Resource Category	Characterization / Examples
Resource Atom	Resources that a priori exist in a resource pool, not result of construction: physical disk, device, machine, network.
Resource Construction • **Resource Aggregation:** - Resource Collection - Resource Container • **Resource Composition or Transformation:** - Virtual Resource - Service Resource	Process of creating new resources from either a priori existing or from prior constructed resources, two categories: resource aggregation and resource composition or transformation. new resource as result of a whole-part relationship with parts are visible and referred to by requestors. group entity for resources: - cluster of machines. container entity that contains resources: - e.g., a UDC farm [UDC]. new resource as result of transformation process with requestors referring to the result, not parts. functionally fully compatible resource: - virtual machine, server partition, virtual network, - virtual disk. resource with new capabilities and properties = service: - file service (file server), database service, web services.

Table 11: Resource classification in resource atoms and resource constructions.

9.3.11 Complex Resource Constructions

Resources are the primary concern of a DCI-OS for management. Due to the diversity of resources found in a data center, a *rich information model for resources* had to be developed for the DCI-OS in order to cope with a very diverse set of resources.

As mentioned earlier, existing management systems in data centers have the restriction to only be capable of dealing with physical resources as they exist in data centers. These systems cannot, for instance, deal with virtual resources as they have become available recently. The rich information model provides comprehensive expressiveness for complex resource compositions using abstractions of resource atoms and resource constructions, aggregation, composition and transformation.

Resource constructions are classified into resource aggregation and resource composition. Resource aggregation falls into resource collections and resource containers. Resource composition is differentiated into virtual resources and service resources. Since both categories are established as result of transformation processes, the term resource transformation is also used for resource composition.

The classification of resources, atomic resources and constructed resources as well as the different categories of resource constructions is summarized in Table 11 and discussed in more detail in the following sections. Resource construction may occur recursively referring to prior constructed resources of any of the categories.

9.3.11.1 Resource Atom

Resource Atom – A resource atom is a resource entity that represents a resource which a priori exists in a resource pool and has not been subject to prior resource construction by the DCI-OS. Resource atoms are the smallest units DCI-OS is aware of. Resource atoms are not further resolvable by the DCI-OS and can only be treated as a whole.

Examples of resource atoms are physical disks, devices, machines, installed networks, etc.

9.3.11.2 Resource Construction

Constructed Resource – A constructed resource is the result of resource construction, the process of aggregating or composing new resources from other resources. Constructed resources are new resource entities with own identity and own representation in the DCI-OS. Constructed resources are classified into resource aggregations and resource compositions (transformations).

Examples of constructed resources are UDC farms, server partitions, clusters, virtual disks, virtual storage volumes, etc.

Resource Construction – Resource construction is the process of creating new, higher forms of resources as result of operation of the DCI-OS by aggregating or composing new resources from more elementary ones, which may either be atomic or prior (recursively) constructed resources.

Constructed resources form new identities, new "whole things" as referred to in UML with new identities and identifiers. Constructed resources are first-class entities with new properties and capabilities and their own representation in the DCI-OS. Constructed resources can establish new contexts within which contained resources exist.

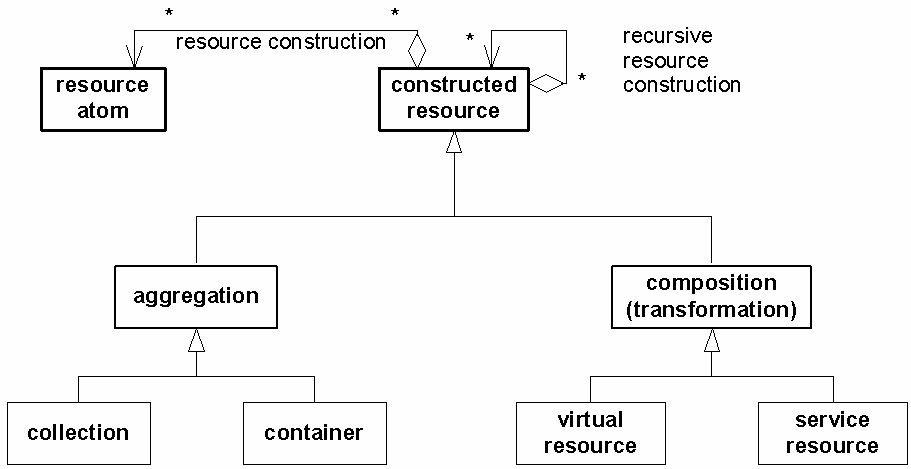


Figure 68: Classification of complex resource constructions.

UML modeling distinguishes between aggregation and composition. Aggregation in UML is a special form of association that specifies a whole-part relationship (or

is_part_of relationship) between the aggregate (the whole) and the parts. Parts of an aggregation can still be visible from outside. When the aggregate is being dissolved, parts may not be affected. And vice versa, when parts are removed from an aggregation, the aggregation continues to exist. Empty aggregations with no parts may exist.

Composition in UML is a stronger form of aggregation with strong ownership and coincident lifetime of the parts and the whole; parts with non-fixed multiplicity may be created after the composite itself, but once created, they life and die with it; such parts can also be explicitly removed before the death of the composite. Parts exclusively belong to only one composition. They are often created and exist only within the context of the composition. Parts are not visible to the outside, only the composition as a whole. When the composition is dissolved, it all its parts are dissolved as well. When parts are dissolved, it has impact on the existence of the composition. Parts cannot exist independently of the composition.

The variety of resource constructions can also be classified in resource aggregations and resource compositions with adopting the terms from UML and slightly modifying them for the purpose of resource construction.

9.3.11.3 Resource Aggregation

Resource Aggregation – Resource aggregation is a resource construction where a whole-part relationship between a set of atomic or prior constructed resources and a new aggregating resource is established. Resource aggregation allows organizing resources as identifiable groups. Contained resources are visible to the outside and can be accessed as they exist from outside. The resource group can exist independently of the (eventually empty) set of contained resources, and resources can exist independently of the group.

Resource aggregation falls in two categories, resource collections and resource containers.

Resource Collection – A resource collection is an entity that represents a group of resources and can be identified as a whole. Resources may belong to multiple collections. Dissolving a collection does not cause dissolving contained resources. Both, resources as well as collections exist as resource entities and exist independently of one another. To the outside, individual resources are primary objects for actors, not collection entities.

An example of a resource collection is a cluster of machines. The cluster exists as resource entity with identity and properties, and each of the machines belonging to the cluster exist as independent resource entity with own identity and properties. Requestors primarily refer to individual machine resource entities, not to the cluster entity which represents the grouping of machines. The purpose of maintaining grouping entities is to provide structural support for management and attaching policy, for instance.

Resource Container – A resource container is an entity that contains a set of resources that is identified as a whole. Resources cannot belong to multiple containers. Dissolving a container does not cause dissolving resources. Both, resources as well as containers exist as resource entities and exist independently of one another. To the outside, the container entity is typically exposed and primarily referred to, not the contained resources.

An example of a resource container is a UDC farm, which is handed over to a requestor as a whole (as a separate, identifiable resource entity farm). Although inner resources are accessible in a farm individually, the primary entity that is referred to is the farm.

9.3.11.4 Resource Composition (Resource Transformation)

Like in operating systems, resources can be transformed from lower into higher resources performing higher functions, having new behavior and new properties. Those resources are established as result of transformation processes performed by a control system. The transformation process uses underlying resources in order to establish a new resource.

Unlike resource collections or containers, only the result, the new transformed resource is exposed to a using entity hiding the underlying resources used for the transformation process. Transformed resources depend on the transformation process, its execution and on the underlying resources without which they cannot exist. Due to this analogy with composition, resource transformations have been classified as resource compositions.

Any kind of a result of a transformation process using resources internally can be classified as resource transformation or resource composition. This includes transformations performed by application processes opening the path for incorporating "software resources" or services into the presented model. Services considered as resources such as file services or data services, or application services used by other applications, fall in the category of resource compositions or resource transformations. Providing services is result of a transformation process controlled by application programs and executed by processes.

Resource Composition (Resource Transformation) – Resource composition is a resource construction that is established as result of transformation processes using underlying atomic or prior constructed resources. The transformation process creates a new resource with new identity and new properties and capabilities. Since composed resources depend on the transformation process, its execution and the resources needed for it, they cannot exist independently of inner resources. Inner resources used for the transformation process are not exposed outside.

Two kinds of resource compositions exist: virtual resources and service resources.

Virtual Resource – A virtual resource is result of a resource transformation process establishing a new, functionally compatible resource to an underlying resource (usually excluding time behavior). Transformation creating a virtual resource involves an indirection providing the control point for the transformation process.

Examples of virtual resources are virtual machines, server partitions, virtual networks, virtual disks, etc. Virtualization transformations are not necessarily performed by, but are under the control of the DCI-OS.

Virtual resources are typically used for resource multiplication or resource partitioning. Virtual resources are also used to impose protection boundaries when resources are shared among requestors providing containment, isolation and protection of resource shares and creating an environment of exclusive use by requestors.

Service Resource – A service resource is result of a resource transformation process and provides a new resource (or service) with a new function, new behavior and properties. Software services are subsumed under service resources.

Examples of service resources are file services (file servers), database services, etc. Any kind of web services can be seen as a service resource in this classification.

After the discussion of the DCI-OS Information Model, the sub-layer of Resource Pool Managers is discussed in the next section.

9.4 The DCI-OS Resource Management Layer

The DCI-OS Resource Management Layer provides the central resource management capabilities of a DCI-OS for the resources in a data center.

It consists of three major parts of the Resource Pool Managers, the Resource Pools and the Resource Pool Drivers. Resource management is organized around a concept of Resource Pool, which contain resource instances of a particular type each. Resource instances are allocated and assigned to Infrastructure Services as needed. Resource Pool Managers maintain not only current resource inventory, but maintain profiles of resource capacity and resource demand over time enabling them to also manage the information about future resource availability and use, which was one of the key requirements for the architecture of the DCI-OS and represents a new key capability.

A second major innovation is the development of a complex Resource Request Format based on which resource sets can be requested, allocated and assigned. This Resource Request Format also expands the capabilities of current resource management systems which only can deal with individual requests for resources of one type, but not complex sets of resources, some of which eventually as result of resource constructions.

As consequence of these innovations, a separation was introduced between resource allocation and assignment, which is explained in more detail in section 9.4.3. While resource allocation is “anonymous”, which means that a certain resource capacity is granted to a requestor, resource assignment refers to specific resource instances which are assigned by the Resource Pool Manager to fulfill an allocation. This principle of late binding helps keeping resource requests isolated form fluctuating resource inventory in a data center, which is particularly important for managing future resource requests. Late binding between resource allocations and assigning resource instances when resources are needed is the third major innovation of the DCI-OS Resource Management Layer and realized in Resource Pool Managers.

Resource Pool Drivers produce the resources for pools, which can be whole or shares of physical resources or generated resources, such as virtual resources, which require explicit creation steps.

9.4.1 Resource Pool Manager

The Resource Pool Manager has two major system components:

- **Allocation System** – decides about (accepts or rejects) requests for resources from requestors and, if accepted, allocates requested resources (quantities or instances depending on the detail of requested resource specifications) accommodating the requested demand for resources. The allocation system memorizes resource allocations. Changes in resource capacity, availability or policy may cause changes in existing resource allocations.
- **Assignment System** – assigns resource instances to (unbound) resource requests. When a requestor has fully specified the details of specific resource instances as part of the request, assignment takes place immediately after the resource request has been received. When the requestor has specified only kinds and quantities of resources, not specific instances, the assignment system can optimize resource assignment for binding resource instances to requests at a late point in time. The

allocation system will reserve respective capacity such that assignment can be done as late as at the time when the resource request has become due.

Run-time deployment and operations control system for assigned resources will be performed by the Deployment Manager which was presented in section 8.2. The Deployment Manager will physically access resources in order to create the desired configurations. Part of the resource-specific configuration is based on requestor's information. At the end of resource deployment, resources are fully configured and ready for use for the requestor. The deployment system will hand resources over to the requestor as final step.

After resources have been handed over to the requestor, the requestor has control over resources for use. The operations control system observes usage of resources used by requestors ensuring that use complies with commitments made by the requestor, and otherwise take corrective action, a principle called policing.

Figure 69 shows the main components of Resource Pool Manager. The requestor (an instance of an Infrastructure Service) submits resource requests. All resources requested in one context are contained in a request context. A request context exists as long as state about requested resource is maintained in the Resource Pool Manager. Resource requests are maintained as request contexts in the request inventory in the Resource Pool Manager.

The allocation system is the first system processing a resource request. The allocation system internally maintains an allocation inventory of allocatable resource specifications. From those specifications, capacity can be allocated for resource allocations accommodating demands from accepted resource requests.

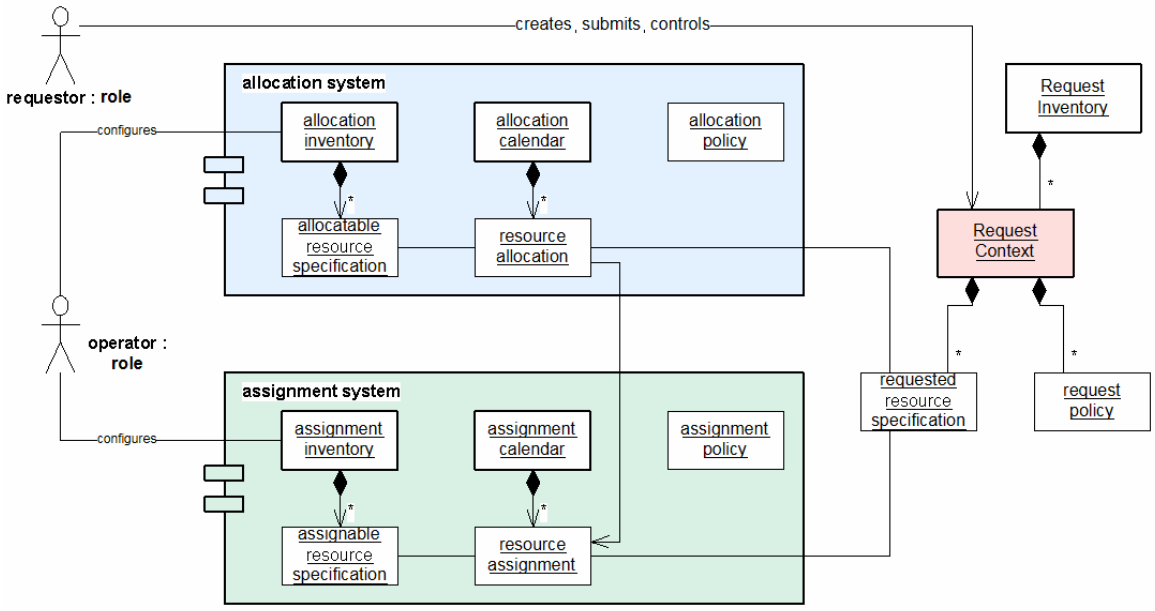


Figure 69: Architecture of the Resource Pool Manager.

The assignment system internally is structured similarly as the allocation system with the difference that specifications maintained in the assignment inventory do not refer to anonymous kinds of resources and available capacities of those resources, but to specific resource instances, which are currently present in the Resource Pool Manager. The

assignment system then also maintains an assignment calendar of those resource instances.

The operations control system is also the contact point for requestors and applications asking for further resources or releasing unneeded resources. The operations control system may also adjust resources transparently for certain applications according to observed use and workload patterns, a principle called flexing.

9.4.2 Resource Request Workflow

The Resource Request Workflow spans across the Resource Pool Managers in the DCI-OS Layer with responsibilities of allocating resource capacity and assigning resource instances and the Infrastructure Services Layer with the requesting Infrastructure Service, where the deployment and control tasks occur as described in chapter 8.

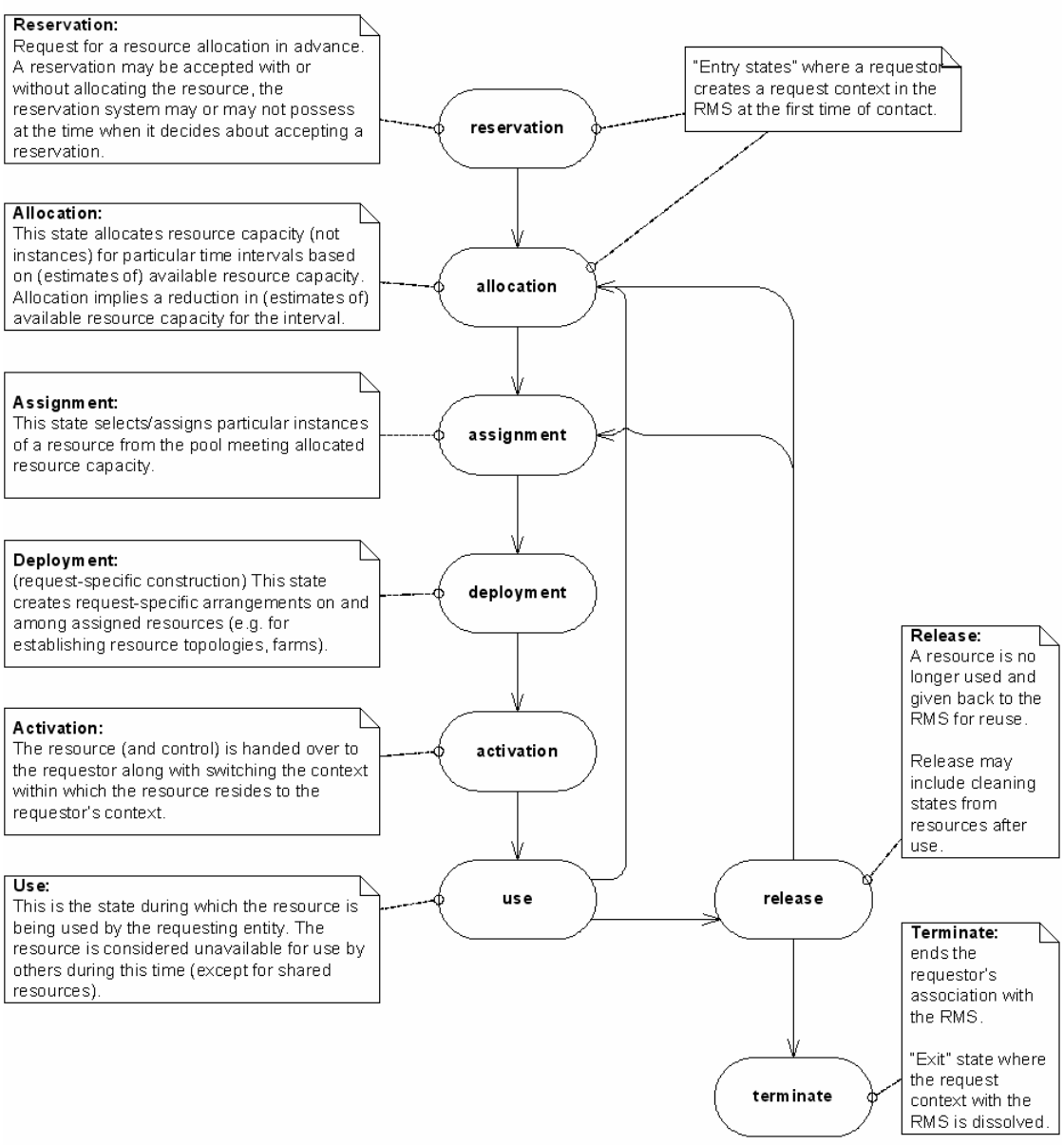


Figure 70: Resource Request Workflow.

The Resource Pool Manager passes each resource request through several stages:

1. **Allocation** – Allocation matches resource demands described as part of requested resource specifications in resource requests with (estimates of) resource capacity described as part of allocatable resource specifications in the allocation inventory. Successful allocation implies a reduction in the available resource capacity for the time intervals for which resource capacity has been allocated.

initiation:	on arrival of a resource request received from a requestor;
input:	request context containing a set of requested resource specifications;
result:	*case accept:* resource capacity has been allocated to each requested resource specification according to requested quantities in requested time intervals; the allocation system memorizes allocations in form of allocation entries in the allocation calendar; *case reject:* no state has changed in the allocation system; the requestor is notified about the rejection of the resource request.

2. **Assignment** – Individual resource instances from an underlying resource pool are selected and bound to accommodate allocated resource capacity. Assignments specified by the requestor are immediately made. Assignments for anonymous resources capacity allocated for a request is assigned shortly before due time. Resources have become due when the earliest time of intended use has been reached. Assigned resources are marked in the assignment calendar and cannot be assigned to other requests for the period of assignment. Resource sharing can be achieved by either configuring multiple entries in the assignment calendar for one resource or can be arranged by requestors outside the assignment system.

initiation:	either on arrival of a resource request when requestor requests specific resource instances, or automatically initiated by the allocation system shortly before resource become due for use;
input:	request context containing a set of requested resource specifications (the assignment system decides which instances from the resource pool are assigned to accommodate requested quantity), requested resource specifications provided by the requestor may refer to particular resource instances (then the assignment system only determines the availability of resources for requested times);
result:	*case success:* resources have been assigned and are bound to requested resource specifications in resources requests; the assignment system memorizes assignments in the assignment calendar; *case failure:* no state has changed in the assignment system; the requestor is notified about the rejection of the assignment request.

3. **Deployment** – Deployment performs request-specific resource configuration and resource construction when resources do not exist. Deployment accesses physical resources. Request-specific resource initialization and configuration is performed based on the information provided in the resource request. Launching a virtual machine is an example of constructing a resource during deployment.

initiation:	shortly before due time; ("shortly" mainly refers to the time needed to complete deployment and activation of resources);
input:	request context containing a set of assigned resource specifications;
result:	*case success:* all resources of the resource request are configured and initialized as specified, constructable resources have been constructed, *all* specified resources are ready to be used by requestor; *case failure (at least one resource could not be deployed successfully):* no state has changed in the deployment system; other deployments associated with the request are rolled back, and the request is rejected with a resource deployment failure.

4. **Activation** – Activation is the state in which a set of deployed resources of a resource request (as an entirety) is handed over to the requestor for use. Activation is performed by the deployment system finalizing resource deployment. Activation implies notifying the requestor about an access path to resources and switching control to the requestor.

initiation:	immediately before due time;
input:	request context with accompanying resources deployed in the resource pool;
result:	*case success:* the requestor is notified with and informed about the access path to resources, the requestor can immediately begin using resources; *case failure:* no state has changed in the deployment system; the request is rejected with a resource activation failure.

5. **Use** – The requestor uses resources under own control. Inside the Resource Pool Manager, used resources (and associated resource requests with specifications) have moved to the operations control system supervising the use of resources.

   The operations control system has authority to preempt resource use, for example when committed time has expired or when "more" resources then announced are used. The requestor may explicitly, or the operations control system may implicitly, suspend resource use making them temporarily available to other purposes. During suspension, requestors' states will be preserved allowing later to resume using the same resource or a resource of the same kind. Requestors should not notice when use has been resumed on other resource instances (unless explicitly excluded by the requestor).

   The requesting entity may request modifications of own resource contexts any time (including run-time) in order to change time of use, quantities or kinds of resources associated with one of its existing request contexts.

initiation:	at due time;
input:	request context with accompanying activated resources in the resource pool;
result:	*use:* the requestor uses resources under own control, the operations control system supervises resource use; *release:* the requestor releases resources permanently (no state is kept); *suspend:* the requestor or the operations control system temporarily suspend

	resources use; the operations control system preserves state of the resource such that the requestor can resume resource use later; *resume:* state has been restored on the resource (same or same kind) such that the requestor can continue using the resource.

6. **Release** – A requestor returns resources permanently. No state is preserved for released resources. Releasing individual resources does not terminate use of other resources. Releasing a resource means that requestors return control over resources back to the operations control system, which will clean the resource from requestor's states and return it to the resource pool. The assignment system is notified about released resources such that those resources can be reassigned.

initiation:	Initiated during use by requestor (e.g. by invoking a *resource_release()* function in the operations control system); or initiated by the operations control system;
input:	request context with set of resources marked to be released;
result:	The requestor can no longer use or access released resources; the operations control system cleans resources from requestor's states and returns them to the resource pool; the assignment system is notified and will reassign them.

7. **Suspend** – Requestors temporarily give used resources back to the operations control system (for instance to reduce charges in pay-per-use regimes). The requestor's state is preserved for suspended resources and later restored when the same resource or a resource of the same kind is resumed.

   In order to enforce policing, or to accommodate sudden unforeseeable resource demands of higher-prioritized requestors, the operations control system may preempts resources from requestors exercising its control over resources. Requestors and their applications may or may not be aware of suspended resources. Applications may slow down or pause when resources are preempted by the operations control system.

   The assignment system must be notified about suspended resources. The assignment system could have been a source to initiate preemption in the operations control systems for accommodating sudden resource demands. The assignment system will also decide about further use of suspended resources.

   Suspending resources is a mechanism to accommodate unforeseeable resource demands reducing the capacity that has to be provisioned for this purpose otherwise increasing average utilization. Suspension practically can only be achieved in virtualized resource environments.

initiation:	during operation by the requestor or the operations control system (preempting resources);
input:	request context with accompanying currently used resources;
result:	Requestor's state on suspended resources is preserved, but resources are unavailable for requestors; affected applications may slow down or pause for the time of suspension; The assignment system is notified about suspended resources and also decides about further use of resources.

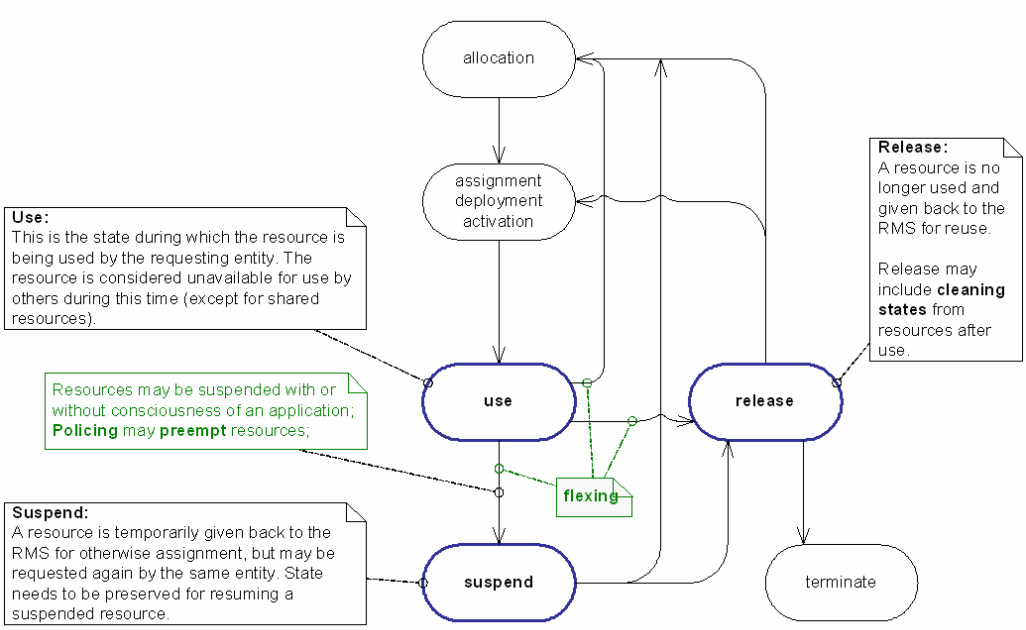


Figure 71: Refined Use-state of the resource request workflow.

8. **Resume** – Requestors may initiate resuming resources by interacting with the operations control system for resources requestors have explicitly suspended; since requestors may not be aware or preempted resources, the operations control system must initiate resuming resources to requestor contexts.

   The same instance(s) or instance(s) of the same kind are resumed with reinstantiated states into a requestor's resource context. The assignment system chooses resources to be resumed. The deployment system reinstantiates states on resources and activates resources.

   Suspended resources can also be released discarding preserved states.

at the end of the suspension period by requestor for explicitly suspended resources, or by the operations control system for preempted resources;
request context with new assignments for suspended resources and associated states to be reinstantiated on resources;
Requestor continues using resumed resources with the same state as at the time when resources had been suspended.

9.4.3 Resource Allocation

The purpose of resource allocation is to commit resources to requestors based on descriptions requestors issue to an allocation system. The allocation system determines whether a resource request is accepted or rejected. Negotiations about requested and committed resources may be performed. Negotiations may involve other requestors, brokers, or other resource allocation systems. When accepted, requested resources are reserved from the allocation inventory for the time periods requested. The allocation inventory maintains information about resources and their available capacities over time. The inventory maintainer role (a person or an automated control system) configures and adjusts resources and capacities that are made available for allocation.

Resource Allocation is the process of committing resource capacity to requests for resources. Committing resources implies setting resource quantities aside from a resource inventory for requested times by reducing available capacity by requested and allocated resource quantities. Committed allocations are memorized in the allocation system as relationships between allocated resources and resource requests, and implicitly their requestors.

An implementation of an allocation system based on the principles presented here is described in [Kön04].

9.4.3.1 Resource Allocation Information Model

The information model describes the major elements of information that are needed inside and outside the allocation system in order to allow it to function.

Following elements have been identified for the resource allocation information model:

- *Allocation Inventory* – maintained inside the allocation system by the inventory maintainer role. It contains information about allocatable resource types with their capacity profiles; it also provides the major information base for exposing specifications of allocatable resources to be used by requestors in form of allocatable resource specifications.
- *Allocatable Resource* – resource that can be requested for allocation including ground resources and views or constructions established upon ground resources. Specifications of allocatable resources (types, prototypes, templates) are published to requestors and used by them for constructing resource requests.
- *Allocatable Resource Specification* – An allocatable resource specification is the information about an allocatable resource published to a requestor role. Various forms may exist such as fully specified type, partially specified prototype or template. Specifications may include type, instance and quantitative information as well as policy. Requestors use allocatable resource specifications for constructing resource requests.
- *Resource Request* – issued by requestors containing information about resources, quantities and the times for which they are requested.
- *Request Inventory* – maintained inside the resource allocation system containing accepted request contexts that currently have state associated in the allocation system.
- *Allocation Calendar* – maintained inside the resource allocation system containing the information about committed resource allocations.
- *Resource Allocation* – maintained in the allocation calendar containing information about one resource allocation as an association between a resource request and a set of resources committed to that request.

A **Ground Resource** is a resource that can directly be requested from subsequent assignment and deployment systems. Thus capacity can be provisioned for ground resources in the allocation inventory in form of a capacity profile.

Ground resources are not necessarily equivalent to resource atoms depending on the capabilities of subsequent resource assignment and deployment systems. Resource assignment or deployment systems may already expose constructed resources (non-

atoms) that are seen as assignable and deployable, atomic resources relatively from the allocation system's perspective. Those resources are considered ground resources, versus resource atoms. A resource atom is a resource entity that a priori exists in a resource pool.

In addition to ground resources, the inventory maintainer role may configure the allocation system to expose higher views or constructions upon ground resources that are more convenient or more appropriate for requestors to use. Establishing and exposing non-ground resources implies that the inventory maintainer role provides all relationships in respective view or construction entities that allow resolving requested resources into ground resources before allocation, a process called resource grounding. Views or constructions do not have capacity associated. When requested, they are translated into corresponding capacities of associated ground resources.

Resource Grounding is the process of translating views or constructions established upon ground resource specifications, which are published and used by requestors, into corresponding ground resources when requested before allocations are made.

A resource allocation represents the relationships between quantities of resource types that are requested by Infrastructure Services and that are committed from Resource Pools. A view is translated into a corresponding quantity of associated ground resources.

An example of a resource view established upon a ground resource is shown in Table 12.

Assoc. ground resource	(cpu="Pentium 4", speed="2300GHz", ram="2GB", disk="120GB")
Resource view I	(architecture="IA32")
Resource view II	(system="XP")

Table 12: Example of two resource views on the same ground resource.

Instead of exposing only the ground resources shown in Table 2, resource view I provides a view of an allocatable system as IA32 architecture without further refining the type of CPU, clock speed, memory size or other parameters. The view entity maintains a reference to the underlying ground resource type that is followed during resource grounding when resources of view I are requested for allocation.

Resource view II exposes the capability that each of the ground resources can also be configured for running the XP operating system. Requestors may request XP systems.

Views allow publishing and requesting resources in more abstract terms making them more independent from changes and undesired detail in underlying resources.

Assoc. ground resource	(cpu="Pentium 4", speed="2300", ram="2048", disk="120")
Resource construction I	(architecture="three-tier", app_server="BEA")
Resource construction II	(architecture="web server farm", server="apache", os="linux")

Table 13: Two resource constructions on the same set of ground resources.

In the example in Table 13, two simple constructions upon ground resources are established. Requesting one quantity of a "three-tier" "BEA" architecture may internally be associated with requesting 4 machines of shown ground resource types for running the

web server, 2 machines for the application server, and one machine for a database. One quantity of the resource construction would translate into 7 quantities of ground resources in this example.

From the perspective of the allocation system, only the 7 quantities of ground resources are associated with the resource request. All further information about specific machine configurations is not considered during allocation and passed on to subsequent assignment and deployment systems as part of the resource request context.

9.4.3.2 Complex Request Format for Resource Topology

A **Resource Request** is the information submitted by a requestor role to an allocation system about a set of resources that are requested within one context. Resource Requests contain requested resource specifications refining published specification of allocatable resources associated with demand profiles.

A resource request contains all resources (and further information) that are jointly requested within one context. The representation of a resource request thus appears in form of a context entity, called resource request context.

A **Resource Request Context** represents the context within which a set of resources is requested containing requested resource specifications along with other information such as about the requestor.

A resource request context persists in the system as long as information about a resource request needs to be maintained (over assignment, deployment, operation and eventually longer for archiving purposes).

As a resource request context travels along its workflow through the Resource Pool Manager, various subsystems will refine contained resource specifications towards finally providing requested resources to the requestor at requested times.

A resource request contains the following information:

- Information about the requestor (identification, authentication information, address, etc.).
- Indication whether the request is an initial (or anchor) request describing an absolute amount of resources requested over time, or whether a request refers to a previous request and describes relative amounts of resources to be added or removed from the referred request.
- Specification of requested resources:
  - reference to resource types, instances, templates exposed as allocatable resources by a Resource Pool Manager,
  - quantities of those resource types, instances, or resource templates with the time-lines over which resources are requested (when, duration), called demand profiles,
  - requestor's constraints associated with requested resources (requestor-specific resource configuration can be provided here).
- General requestor policy information associated with the request and to be obeyed by the Resource Pool Manager.

- Structural information how requested resources are connected in form of topology information upon requested resource specifications.

The structural break down of a resource request can be derived from its basic information. Three major components exist in the Complex Request Format for Resource Topology:

- requestor information,
- set of requested resource specifications (RRS) as quadruples of:

  RRS = { (resource type/instance/template,
  demand profile,
  anchor_ref,
  constraints) },

  with:

  - resource type, instance or template are referred to in the requested resource specification (e.g. reference to a CIM resource type); resource types, instances or templates are requestable when they are exposed by a Resource Pool Manager as allocatable resource specifications (see Allocation Document),
  - the demand profile defines demanded resource quantity over time,
  - reference information can point to an associated resource in an anchor request for incremental flexing (to add/remove quantity to an existing profile in the anchor request),
  - constraints may apply to a resource specification, or to individual parts of profiles.
- A requestor can provide policy information that applies to the request in general.

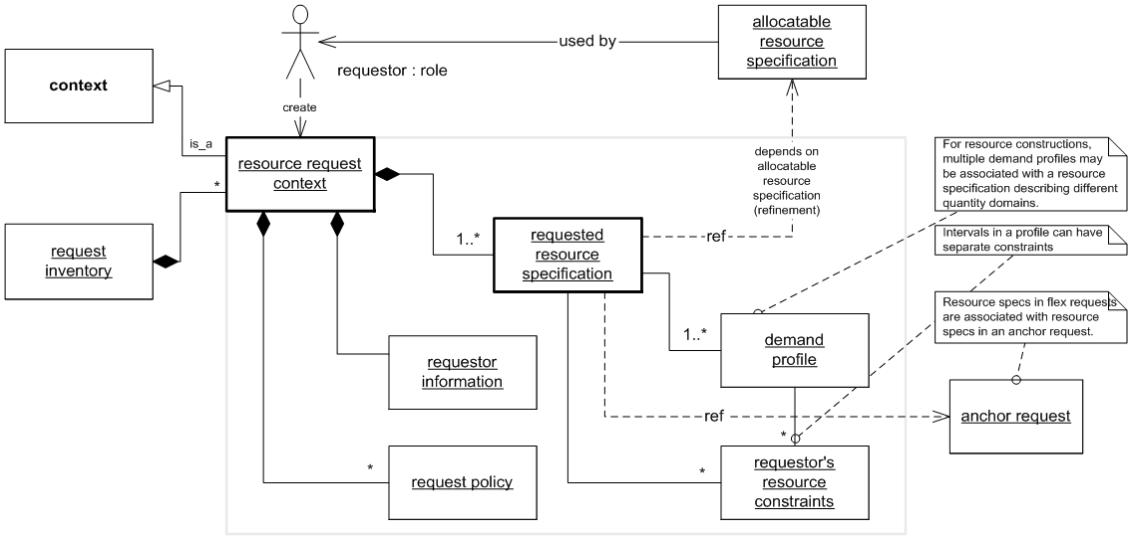


Figure 72: Resource request for a Resource Topology with external dependencies.

Figure 72 shows the structural view to a resource request with its external dependencies. Resource requests are contexts. A requestor role (user or system) refers to allocatable resource specifications exposed by a Resource Pool Manager in order to create a resource request. Every allocatable resource can in principle be requested by including a reference

to it in the request and associating a demand profile specifying what quantity of the resource is demanded over time.

Constraints may be specified for a requested resource specification or, at a finer grade, for periods (intervals) in a profile. Each requested resource specification can refer to a resource specification of a referred anchor request for incremental flexing by changing the demand profile in the anchor request.

9.4.3.3 Resource Type, -Instance and -Template

Resource types, instances or templates are referred to by requested resource specifications in combination with demand profiles indicating the quantity requested over time. A Resource Pool Manager only understands resource types, instances or templates which are known to the Resource Pool Manager and exposed to requestors by its allocation inventory.

A Resource Pool Manager can expose three kinds of resources as allocatable resources:

- fully specified resource type specifications (e.g. CIM type for machines of type lpr2000),
- fully specified instance specifications (e.g. individual machine l024.hp.hp.com), or
- fully or non-fully specified template specifications of resource types or instances, which are to be refined by a requestor when submitted.

As allocatable resources are understood resources that can be requested (referred to in resource requests). Virtual resources or other resources that do not a priori exist, but can be instantiated or constructed when requested or will be available when requested in future, can be configured as allocatable resources in the Resource Pool Manager's allocation inventory.

Resource types can be as simple as shown (such as machine of type lpr2000). Resource types may also define complex constructions such as n-tier resource architectures.

When simply quantities of resource types are requested, only a reference to the resource type definition will be included in the resource request specification. The Resource Pool Manager will allocate resource capacity to the request and assign resource instances at due time in order to fulfill the request.

When a requestor seeks specific resource instances, instance information must be configured as allocatable. Also in this case, only a reference to the allocatable resource instance information will be included in the resource request specification.

In both cases, requestors do not require changing allocatable resource types or instance information. They are referred to as they are in resource request specifications.

In order to allow requestors to customize (extend, refine, alter, construct) resources from existing allocatable resource specifications (types or instances), these basic resource specifications can be used to extend, refine, alter or construct new resource specifications, which can then be included in a resource request.

Allocatable resource specifications may contain constraints defining which extensions, refinements, alterations, or constructions are allowed and accepted by the Resource Pool Manager when constructed and requested by a requestor. The requestor must obey those constraints for building valid requests.

Often used constructions (such as n-tier architectures where typical web applications can be deployed) can be provided as templates for constructed, allocatable resources and used by a requestor.

9.4.3.4 Resource Profile

A resource profile defines a quantity of a resource over time. Resource profiles are applied in two forms in the Resource Pool Manager:

- **Resource Capacity Profile** – specifying an available resource quantity in the allocation inventory.
- **Resource Demand Profile** – specifying a requested or demanded quantity of a resource in a resource request, and

Both kinds of profiles employ the same model, operations and data structure. They differ in interpretation in different contexts. For this reason, discussion is generalized to a notion of a profile, which is used as demand profile in the context of resource requests.

The model of a resource profile provides the foundation for expressing demand or capacity as functions of quantity over time. Profiles are fundamental parts of allocatable resource specifications in resource requests. Main goal of the resource profile model was to accommodate a variety of requirements occurring in various use cases of formulating resource requests.

Since resource profiles describe quantity over time, the two axes (time and quantity) are discussed and formalized separately.

As requirements regarding expressiveness for the time axis have been considered:

- periodic reoccurrences (to describe recurring quantity patterns),
- independence of absolute time allowing to use demand profiles of applications at different times,
- means to describe simple profiles in simple terms (e.g. only quantity expressed, no time profile specified – "5 machines" (implicitly means: now and for infinite time)),
- superposition of periodic reoccurrences (nesting periods),
- notion for infinity ("forever"),
- notion for a final termination.

As requirements regarding expressiveness for the quantity axis have been considered:

- absolute numbers (e.g. demand 10 machines, 5TB storage, 1.5GB/s bandwidth),
- range flexing (e.g. demand 5-10 machines, 3-6TB storage),
- statistical distribution of (demanded or available) quantity (e.g. it has been observed that an application demands 5 machines with 70% probability and 8 machines with 30% probability).

These requirements translate into the design of a resource profile that is subsequently described.

9.4.3.5 Specifying Time in Profiles

Profiles define quantity (demanded quantity in resource requests, or available quantity in resource allocation inventories) over time. Time and the specification of time (points, durations) are thus fundamental for profiles.

Absolute time. Various standards exist for expressing absolute time (points). One common standard is ISO 8601:

> Complete date plus hours, minutes and seconds:
>
> YYYY-MM-DDThh:mm:ssTZD (eg 2003-07-16T19:20:30+01:00)

UTC (Coordinated Universal Time) addresses time zones by adding a time zone offsets to GMT.

> 2003-11-05T08:15:30-05:00 corresponds to November 5, 2003, 8:15:30 am, US Eastern Standard Time (-05:00 time offset to GMT).

Relative time. Relative time (duration) has recently been addressed by XML Schema. XML Schema adopts ISO 8601 for its time specification and extends it for relative time durations (http://www.w3.org/TR/xmlschema-2/#durations):

"The lexical representation for duration is the [ISO 8601] extended format PnYn MnDTnH nMnS, where nY represents the number of years, nM the number of months, nD the number of days, 'T' is the date/time separator, nH the number of hours, nM the number of minutes and nS the number of seconds. The number of seconds can include decimal digits to arbitrary precision." The leading P indicates a period (as duration).

For example, to indicate a duration of 1 year, 2 months, 3 days, 10 hours, and 30 minutes, one would write: P1Y2M3DT10H30M. One could also indicate a duration of minus 120 days as: -P120D.

Combining times. Besides specifying points and durations of time, a profile requires defining various sequences of intervals, recurrence, repetition, etc.

Unix cron specifications show one way of combining various intervals and points in time for defining when cron jobs are to occur (following text is taken from http://www.gnu.org/software/gcron/specification.html).

The basic file format consists of lines with 6 fields separated by any number of whitespace characters (excluding newlines). The first five fields describe the times in which each job is to be executed.

1. Minute [0,59]
2. Hour [0,23]
3. Day of the month [1,31]
4. Month of the Year [1,12 | see below]
5. Day of the week [0,7 (where 0 and 7 are Sunday) | see below]

The fourth field, Month of the Year, may also be three letter abbreviations of the months. The abbreviations must only be the first three letters of the month. The fifth field, Day of the Week, also can be three letter abbreviations. The abbreviations may only be the first three letters of the days of the week.

The sixth field is the command that is too be executed should cron determine a match with the 5 previous fields. The sixth field is no longer space delimited so that no special consideration need be taken by the user to pass arguments to the command successfully.

Each of the first five fields may be an asterisk (denoted a match for all possible values), an element, or a list of elements delimited by commas. An element may be a single number, a range of numbers, or a ranger of numbers followed by a '/' and a step value. A range of numbers is denoted by a number followed by a hyphen followed be another number and are inclusive values. Step values tell cron to skip certain numbers in the range given. A step value may also be applied to an asterisk, in which case an asterisk is treated as a range value encompassing the full range of values allowed in that field.

Examples are:

- 1,3-6,10 – matches the values 1,3,4,5,6,10,
- 1-20/4 – matches the values 1,5,9,13,17,
- 2-10/2,5 – matches 2,4,5,6,8,10.

Similar expressiveness of times, intervals and periods is needed for resource profiles. The following section introduces a powerful model for time that is used in resource profiles. The main capabilities that go beyond what can be specified with cron or Outlook are:

- periodic reoccurrences of arbitrary complex series of intervals,
- shiftable interval series allowing to reuse profile (of applications) later when application run later again,
- recursive nesting of interval series,
- half-open intervals (one side of an interval is unspecified, allowing to deal with infinity or unspecified times),
- potential to include events to start or end half-open intervals.

9.4.3.6 Time Model for the Resource Profile

Figure 73 illustrates the time model for a resource profile.

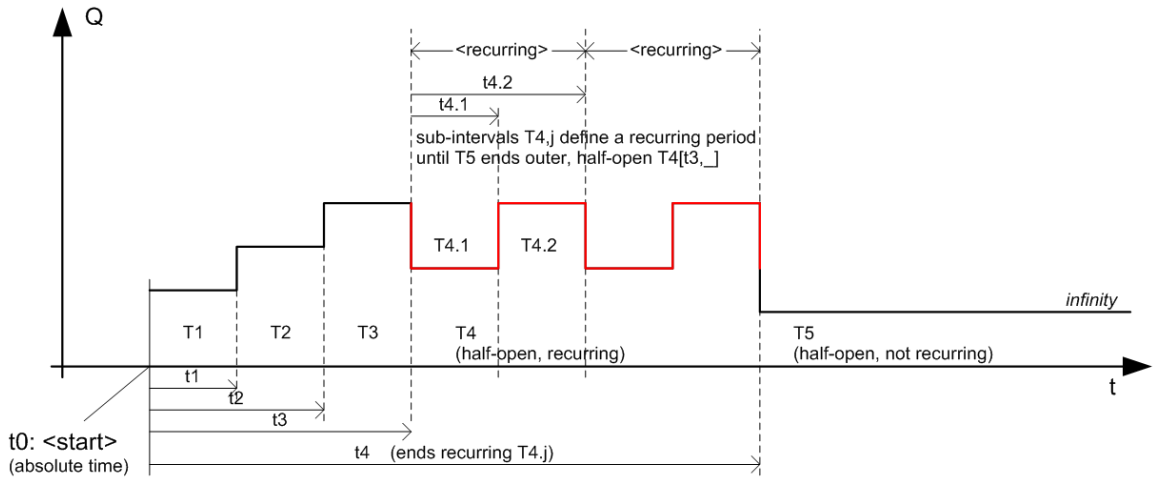


Figure 73: Time model for the resource profile.

Following rules apply to the time axis for specifying time in profiles:

- the time axis of a profile consists of an ordered set of interval definitions T_i (i=1,2,…, n),
- intervals T_i are ordered by start time, intervals with same start times are not allowed,
- each T_i has a quantity Q_i associated (various means exist to specify Q,
- t_0 defines an absolute point in time where the first interval of a profile starts,
- it is recommended to specify all further points t_i relative to t_0 (allows to time-shift the profile),
- an interval T_i is defined as time between t_{i-1} and t_i: $T_i = [t_{i-1}, t_i]$, (i=1,2,…, n),
- half-open intervals omit either t_{i-1} or t_i: $T_i = [t_{i-1}, \_]$ (only start point defined), $T_i = [\_,t_i]$ (only end point defined),
- the unspecified boundary of a half-open interval (_) is implicitly defined by the closing boundary of the closest neighbor interval, if no neighbor exists, infinity is assumed,
- overlapping intervals can be specified, their interpretation is subject to policy,
- each T_i can contain a sub-specification of a series of sub-intervals (defined in a nested, self-similar profile structure),
- the enclosing T_i can specify that the inner interval series is recurring n times or until T_i ends,
- each inner $T_{i,j}$ can have further inner interval series allowing arbitrary deep nesting (or superposition) of discrete, periodic interval sequences,
- a strict order is defined over $T_i = [t_{i-1}, t_i]$ and all inner $T_{i,j} = [t_{i-1,j-1}, t_{i,j}]$, (i=1,2,…, n, j=1,2,…, m).

Instead of recursively describing sub-intervals, periodicity can also be expressed as mathematical function. This specialization is not further discussed since it does not fundamentally change the model. It can be introduced when reason occurs.

Another generalization from time-bound intervals is defining interval boundaries by general events. Events can end half-open intervals, for instance. Expiration of a time can be seen as a form of an event triggering the end of an interval. Other events can also end intervals. Since events are typically not be foreseeable, event-based interval termination will make it difficult to plan and schedule resource use and has for this reason been excluded.

9.4.3.7 Specifying Resource Quantity for the Resource Profile

"Quantity" (Q) of a resource can be described in various ways. In order to incorporate statistical demands, quantity can be modeled as a Probability Mass Function (PMF). Each interval Ti can be assigned a separate function Q_i which may be a simple constant, a range or a PMF (see Table 14).

Four variations are considered for expressing resource quantity assigned to intervals T_i:

Quantity Specification	Example
Constant value: *n*.	5 machines.
Range flexing: *[n,m]* initially *k, n≤k≤m*.	5 – 10 machines, initially 8.
Discrete statistical distribution of n resources as tuples: *[n,p(n)], n=0,1,2…N, Σp(n)=1*.	1 machine requested with probability 0.3, 2 machines with probability 0.6, 3 machines with probability 0.1.
Statistical distribution as probability function.	Gaussian normal distribution with expectancy and standard deviation of a demand.

Table 14: Quantity model for the resource profile.

9.4.3.8 Resource Capacity Profile

Resource Capacity is the current or anticipated quantitative availability of a resource (type or instance). Capacity may be expressed in metrics of continuous ranges or numbers of instances.

Examples of capacity expressed in terms of numbers of instances are: 100 machines of type A or 20 devices of type B. Examples of capacity expressed in terms if continuous ranges are: 1GB/s bandwidth or 10 TB storage.

Capacity Profile – Capacity profile of a ground resource is a function of anticipated capacity of a resource over time.

Only ground resources have capacity provisioned. Capacity of a resource is a function over time about which quantity of a resource type is available (or is expected to be available) over a time period. Capacity profile is provided by the inventory maintainer role.

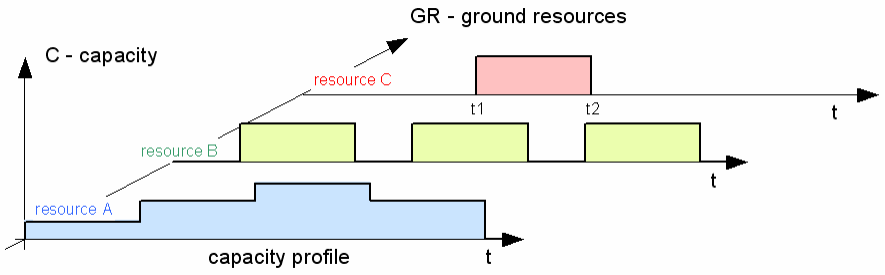


Figure 74: Capacity profiles of ground resources.

Figure 74 shows three capacity profiles for ground resource types A, B and C. Resource types A and B are provisioned with varying capacity over time. Resource type C is provisioned with constant capacity that is only available between t_1 and t_2.

9.4.3.9 Resource Demand Profile

Resource requests are constructed by requestor roles based on allocatable resource specifications. Resource requests are submitted to the allocation system.

Resource Demand is the current or anticipated quantitative use of a resource. Demand may be expressed in metrics of continuous ranges or numbers of instances.

Examples of demand expressed in terms of numbers of instances are: 87 machines of type A or 12 devices of type B. Examples of demand expressed in terms if continuous ranges are: 0.86 GB/s bandwidth or 8.4 TB storage.

Demand Profile of a requested resource is a function of anticipated demand of a resource over time. The Demand Profile is provided by the requestor role. Besides other information, the main information in the resource request context is the set of requested resource specifications referring to published specifications of allocatable resources. Requestors may refine those specifications or construct requested resources from allocatable resource specifications.

Requested Resource – is the result of the instantiation and binding process of a requested resource specification to an either existing or constructed resource in the Resource Pool Manager. A requested resource finally is handed over to the requestor for use at the requested time.

A requestor's goal is obtaining access to requested resources. Requestors express their needs for requested resources in form of requested resource specifications that are submitted to the allocation system as part of resource requests.

Requested Resource Specification – is derived from an allocatable resource specification associated with a requestor's demand profile for a resource.

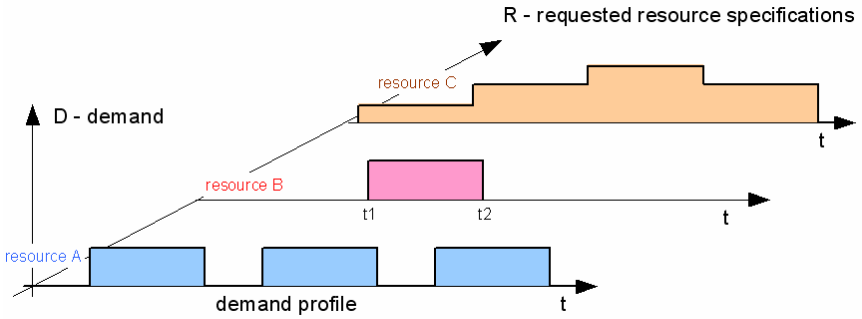


Figure 75: Resource demand profiles.

Figure 75 shows requested resource specifications of three resources with three resource demand profiles representing the combined demand for a Resource Topology.

9.4.3.10 Request Inventory

All requests received from requestors are maintained in the request inventory in the Resource Pool Manager, initially in the request inventory of the allocation system. Multiple request inventories may be maintained in the Resource Pool Manager. At one time, one resource request can only reside in one request inventory, but may be referred to multiple times. This constraint ensures consistency of resource request contexts.

The *Request Inventory* contains all resource request contexts that currently have state associated in the Resource Pool Manager.

When multiple request inventories are maintained in the Resource Pool Manager, a resource request context may transition through these inventories, for example, reflecting the resource request workflow through allocation, assignment, deployment, operation (and eventually archiving).

9.4.3.11 Allocation Calendar and Resource Allocation

The *Allocation Calendar* contains the information about all committed resource allocations. The allocation calendar is the main data structure in the allocation system.

A *Resource Allocation* is the association between a resource request and a quantity of ground resources committed to that request according to the demand profile. Allocated resources reduce available resource capacity.

Within the allocation calendar, two major data structures are maintained: the allocation schedule for each ground resource that correlates the capacity profile of the ground resource with existing allocations, and the resource allocation entry which represents one continuous fraction from a demand profile of a resource request.

A *Resource Allocation Schedule* is a data structure inside the allocation calendar that correlates the capacity profile with current allocations for each ground resource.

A *Resource Allocation Entry* is a data structure inside a resource allocation schedule that represents one continuous interval of resource quantity from the demand profile of a requested resource.

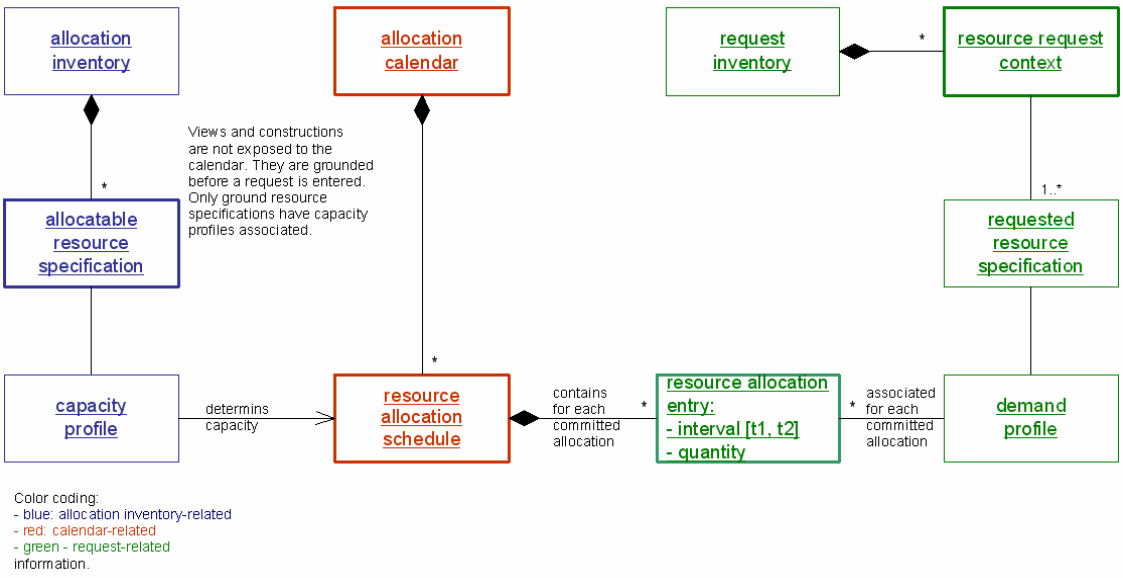


Figure 76: Data structures of the allocation calendar.

Figure 76 shows that for each allocatable ground resource, the allocation calendar maintains an allocation schedule based on the capacity profile of that allocatable ground resource. Each committed allocation causes a number of allocation entries in that schedule representing the demand profile of the requested resource.

The following figure shows the associations between allocatable resources maintained in the allocation inventory, the allocation calendar with allocation schedules for each

allocatable resource, and resource requests maintained in the request inventory with requested resources and associated demand profiles

Figure 77 shows the relationships between the capacity profile for a ground resource (bottom), a demand profile for a requested resource (top), and the resulting allocation as entries in the allocation schedule of the ground resource (middle).

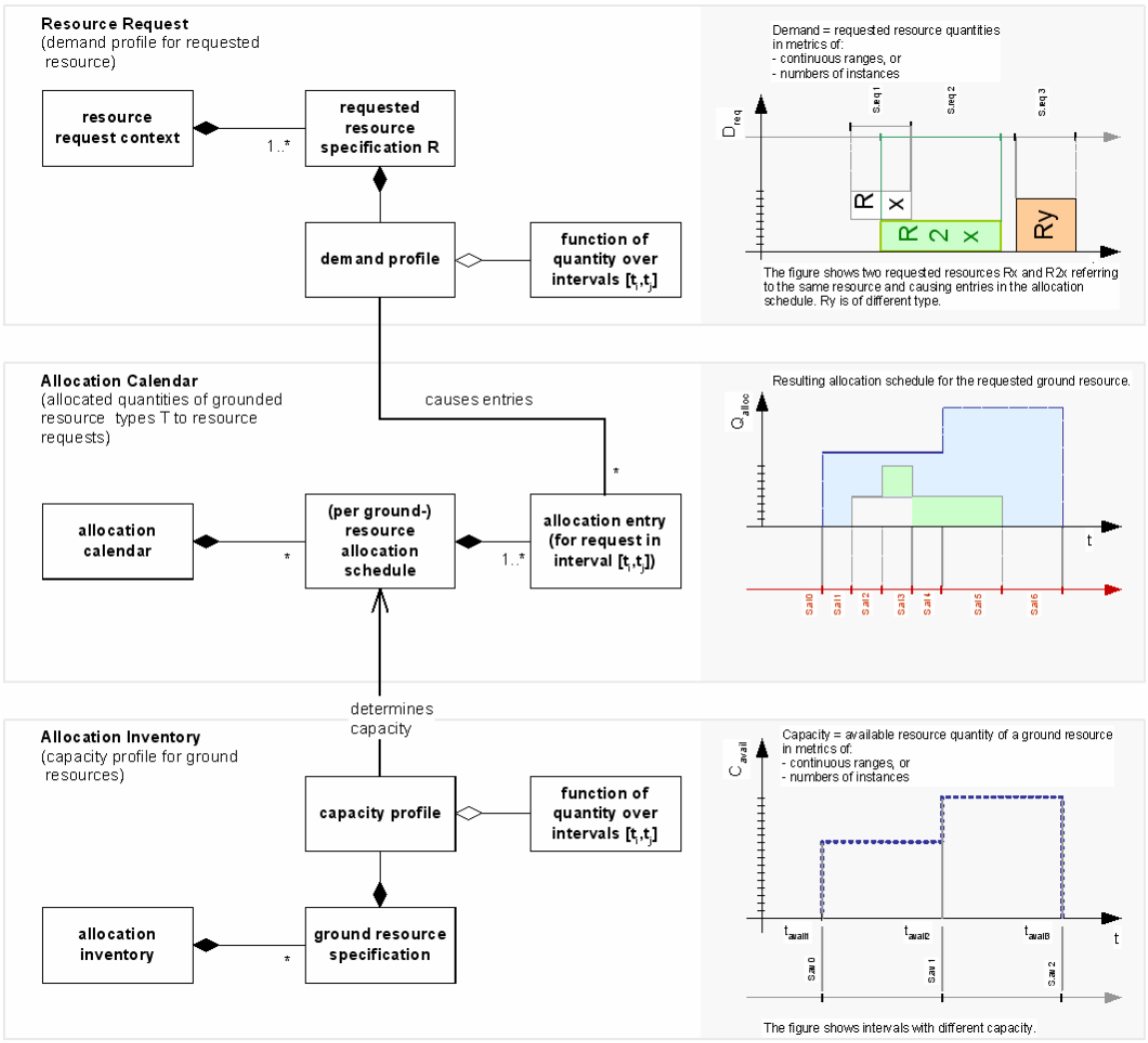


Figure 77: Relationships between capacity and demand profiles.

9.4.3.12 Resource Request Workflow

The internal flow of a resource request is shown in Figure 78. It is assumed that multiple of such flows are performed simultaneously in the allocation system.

As the first stage, resource grounding translates requested resource specifications in ground resource specifications by associating requested views or constructions with ground resources and translating requested quantities.

The validity of resource specifications is verified next. Validity refers to whether all requested resource specifications correspond to allocatable resource specifications. When requested resource specifications such as resource types are not found in the allocation inventory, the request is rejected.

The admission control stage verifies whether resource requests in general are accepted or not, or whether quota apply to requestors. Admission control explores requested demand profiles and validates them with quota.

During the detailed request validation, requested quantities from demand profiles are verified against available resource capacity in the allocation schedule of the associated resource. If the sufficient capacity is available, the requested quantity is allocated in the allocation schedule.

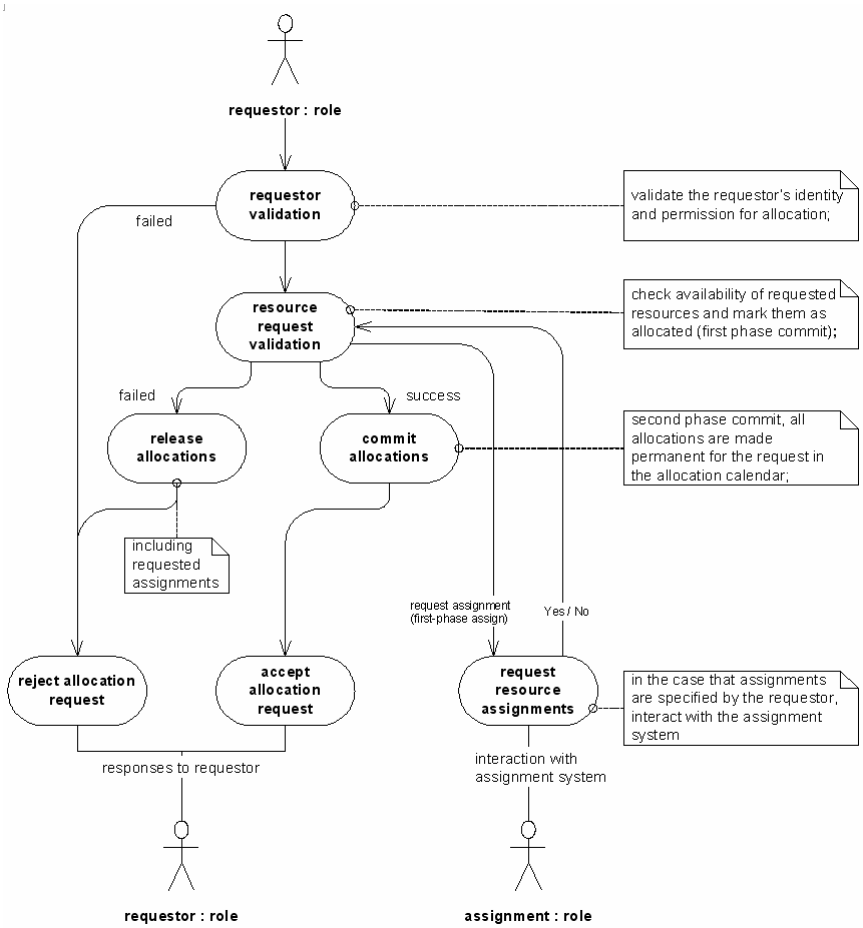


Figure 78: Flow of resource request through the allocation system.

A two-phase commit protocol is used to (first-phase) allocate all resources as requested when sufficient available capacity is found in the allocation schedules. At the end of iterating through all demands in demand profiles of all requested resources, all resources that had been allocated during the iteration are in first-phase allocation state. If the allocation request as a whole can be granted by the allocation system, allocations are made final (second-phase commit), and the allocation request is returned with success.

Assuming an *all-or-nothing* policy, the allocation request fails when at least one part of a requested resource specification could not be honored (properties or demanded quantities). In this case, all prior first-phase allocations are released, and the request returns as failed.

Since instance-related information can be published as part of allocatable resource specifications, requestors may request specific instances. When instances are requested,

respective quantities are marked as allocated in the allocation schedule of the resource. In order to succeed, the assignment system must be contacted whether requested assignments can be made or not. If assignments cannot be made, the request fails. If assignments can be made, the assignment system marks resources as (first-phase) assigned to the request. When the request as a whole succeeds, all of its (first-phase) assignments are committed and finalized. When the request as a whole fails, all of its (first-phase) assignments are released.

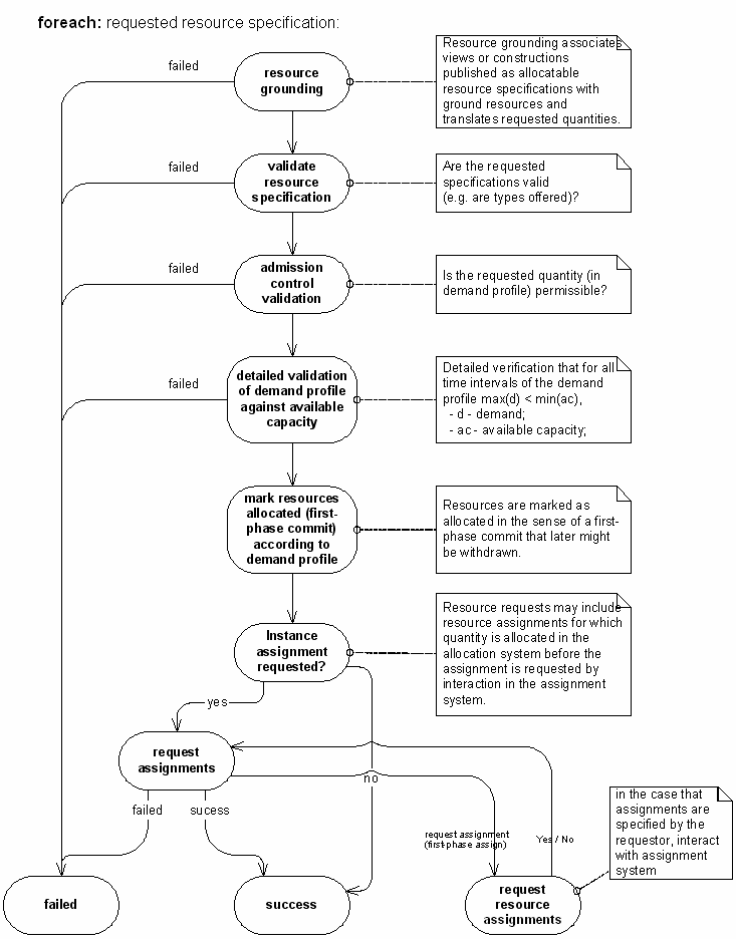


Figure 79: Detailed flow of resource request validation step from Figure 77.

After requestor validation where identity and authenticity of a requestor is determined, resource request validation verifies whether the request can be allocated or not. During resource request validation, interactions with an assignment system may be necessary when instances are requested rather than anonymous quantities of resources.

Figure 79 refines the resource validation stage. The flow in Figure 79 describes the iteration for each requested resource specification contained in a resource request.

9.4.4 Resource Assignment

Resource allocation deals with "abstract" resources in a sense that only quantities and types of resources are considered as information.

Resource Assignment considers specific instances which need to be chosen from resource pools and made available fulfilling a Resource Allocation.

Once resource requests near their commission time, individual instances are selected from resource pools based on some criteria. Eventually those resources need to be created, if they are of a complex (constructed) type.

Resource assignment hence is a section problem of resources from pools. Availability of sufficient quantities of resources at the time of assignment has been ensured by the preceding allocation discussion, which was introduced in section 9.4.3.

9.4.4.1 Resource Scheduling in the Data Center

Since resource assignment here is seen as a selection problem of resource instances from resource pools, it can be compared with *Scheduling* as a known principle from operating systems. Scheduling refers to data center resources such as server resources, network connections and storage. While scheduling in operating systems has been fully automated, it is mainly a manual task in data centers today. People decide about assignments of resources to applications. There is one domain, where automated schedulers are used in a data center context, which are machine schedulers for server clusters used in cluster or high performance computing. Schedulers are also a main component of the Grid management infrastructure [Fost98], [Fost03]. A number of Schedulers exist for Grids, such as the open source Portable Batch System (PBS) [PBS], or the Maui Scheduler [Maui], or Condor [Cond96]. Commercial Grid schedulers are Platform's Load Sharing Facility (LSF) [LSF] or Sun's Grid Engine [N1].

These schedulers can only manage current resource inventories in data centers. Another limitation of Grid schedulers is that those only manage one resource type: servers of some homogeneous type (same processor architecture, same operating system, etc.).

Some Grid schedulers take other resources into account such as network bandwidth, but not the entirety and variety of resources needed for enterprise applications.

9.4.4.2 Resource Assignment Optimization for the Data Center

The resource assignment problem (as a selection problem) was explored in research as an optimization problem to identify, for example, resource sets to be assigned to the same Infrastructure Service that were logically "close". There is an observation that resource sets which are used together show workload and use patterns that are similar to the phenomenon of *Working Sets* [Den68] and *Locality* observed and explored in operating systems.

The approach to working sets and locality in operating systems had to be adopted to the data center environment where, for instance, locality of resources used together in a context typically translates into minimizing network distance in some measure such as numbers of switch- and routing points through which traffic has to flow when crossing from one resource to another, e.g., when a server generates IO traffic to a disk which is connected through a SAN switch residing in an external disk array. In this case, the disk

resources should be assigned for that server from a disk array which is closest to the server.

Manual or simple heuristic approaches to resource assignment work when all resources are equivalent (e.g., servers in a cluster), or when the resource pool is small. With complex topologies, it is possible to create bottlenecks in the shared resource sets resulting in failure to meet application requirements even when sufficient capacity is available in the data center. The assignment system automates selection of the servers within the data center fabric for deployment of the application.

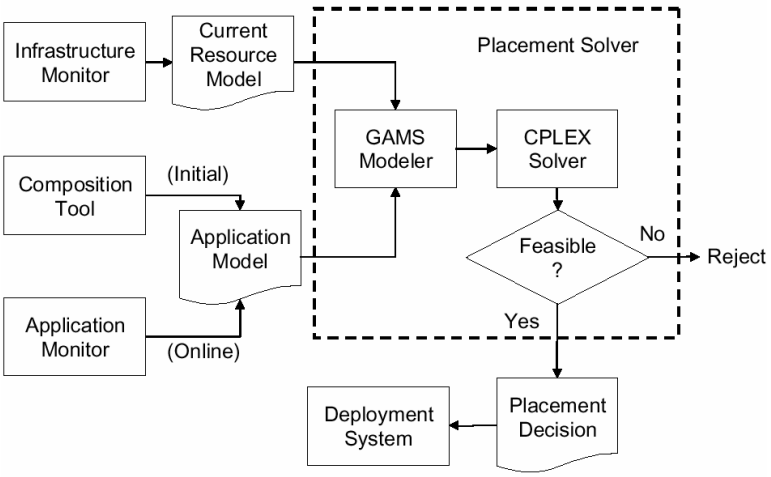


Figure 80: Resource assignment process using an optimizer.

Figure 80 shows one process using a CPLEX solver that was explored to avoid bottlenecks in the connection fabrics and enforcing closeness of resources which are assigned together. The solver requires two models as input: a resource model and an application model. The resource model describes the fabric topology and the required resource capacities. The application model defines the application topology and its resource requirements. The infrastructure monitor tracks the resource inventory, including the connection topology and available capacity. The monitor maintains an up-to-date model of the current state of the environment using information from the resource inventory and monitoring tools.

In this case, the constraints and the objective function required by the solver are dynamically generated from modeled information using the modeling language GAMS, and fed into the CPLEX solver. The latter checks the feasibility of the problem, and finds the optimal solution among all feasible solutions. The detailed models required by the assignment solver are described in [Zhu03].

Substantial work has been conducted as part of the research program and demonstrated in publications by other researchers than the author such that this work is referred here, but not described in detail. The reader is referred to publications such as *Optimal Resource Assignment in Internet Data Centers* [Zhu01], *Resource Assignment for Large Scale Computing Utilities* [Zhu03], *Policy-based Resource Assignment in Utility Computing Environments* [San04], *Extension of the Resource Assignment Problem: Assigning Multiple Application Components to a Single Server in a Generalized LAN Tree Topology* [Zhu04] and *RAMP-A Solver for Automated Resource Assignment in Computing Utilities* [San05].

9.5 The Data Center Component Layer

Data center components comprise the physical environment in a Data Center which is under the control of the DCI-OS. For purposes of management, components expose interfaces, in form of user interfaces (consoles) or programmable interfaces allowing configuration and control via a network.

The DCI-OS Resource Pool Drivers represent software modules in the DCI-OS environment (executing on dedicated management machines), which interact with associated components in the managed environment via programmable APIs over a management network for applying configurations as well as issuing control instructions. In reverse direction, the managed component reports back monitoring data and events indicating state changes in the component.

From the DCI-OS perspective, the Data Center Component Layer represents the devices by their management interface endpoints to which component drivers connect.

Over the years, a number of standards have been proposed and developed for unifying former proprietary interfaces for management. Those need to be taken into account in the DCI-OS Data Center Component Layer.

9.5.1 Management and Discovery of Data Center Components

A number of standards have emerged over the years for unifying protocols and data models for managing and discovering managed components.

9.5.1.1 Management of Data Center Components

Management generally refers to the ability to configure and control managed components and receive monitoring data and notifications about state changes instantaneously.

The Simple Network Management Protocol (SNMP) [SNMP] is a UDP-based network protocol. It is widely used in network management systems to monitor devices for conditions. SNMP is a component of the Internet Protocol Suite as defined by the Internet Engineering Task Force (IETF). It consists of a set of standards for network management, including an application layer protocol, a database schema, and a set of data objects

The Structure of Management Information (SMI) was defined in RFC1155, and the data structure of SMI which called Management information base (MIB) was defined in RFC1156. And IETF defined the SNMP version one in RFC 1157; it has already been the standard of TCP/IP network management in practice.

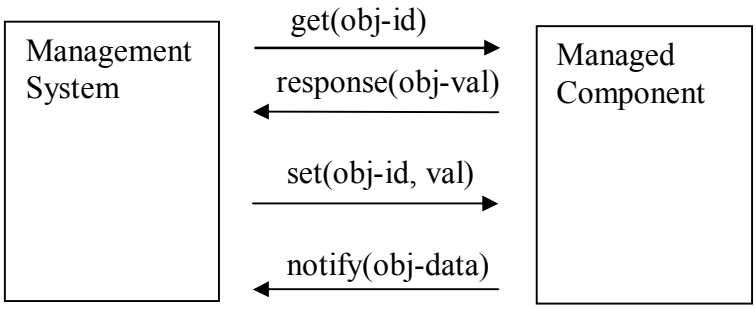


Figure 81: Interaction between management system and managed component.

Figure 81 shows the two basic interaction patterns used in SNMP between the management system and a managed component. The management system can request information from the managed component by issuing a get-request specifying an object in the managed component and the managed component responds with the value of that object.

Control functionality over the managed component is unified into an abstraction of managed objects, a data structure which carry state reflecting the actual state of the component with regard to a certain aspect modeled by the managed objects. Control is exercised by setting the state of the object value.

Furthermore, a managed component can issue a notification (called trap in SNMP, also called event in other frameworks) informing the management system about a sudden change in object state. The management system must be prepared at any time to receive and process incoming traps from managed components.

This data-oriented approach allowed a simple interface with only three basic interactions (get, set, notify) as an alternative for "rich" interfaces exposing each capability by a separate method. The clear advantage of a simplified interface significantly helped standardization efforts. The complexity management interfaces suffered implementing rich interfaces, however, was not resolved but rather moved into the data/object structures referred to by the get, set and notify operations.

Addressing the complexity in the data/object structures hence became the focus for standardization efforts for interfaces of managed components.

Initially, the Management Information Base (MIB) was defined in RFC1156 in 1990. Objects were arranged hierarchically such that they could be identified by a path through the object tree. Since storage was limited in managed components, bits were used to encode object values. The Abstract Syntax Notation One (ASN.1) [ASN1] was introduced to provide a syntactical framework to describe data structures for representing, encoding, transmitting, and decoding object data. ASN.1 is a joint ISO/IEC and ITU-T standard, originally defined in 1984 as part of CCITT X.409:1984. ASN.1 moved to its own standard, X.208, in 1988. The substantially revised 1995 version is covered by the X.680 series. The latest available version is dated 2002.

Later, as more storage became available on managed components, richer descriptions could be used to represent objects and object states. Today, XML-based representations are used by the WBEM [WBEM] standard for management as well as by Web Services-based Management Standards such as WS-Management [WS-MAN].

The Common Management Information Protocol (CMIP) [CMIP] is a protocol for network management. It provides an implementation for the services defined by Common Management Information Service (CMIS). CMIS is part of the Open Systems Interconnection (OSI) body of network standards, allowing interaction between management systems and managed components. CMIS/CMIP emerged from the ISO/OSI network management model and is defined by the ITU-T X.700 series of recommendations.

CMIS defines following basic management operations on managed components:

- M-CREATE - Create an instance of a managed object,
- M-DELETE - Delete an instance of a managed object,

- M-GET - Request managed object attributes (for one object or a set of objects),
- M-CANCEL-GET - Cancel an outstanding GET request,
- M-SET - Set managed object attributes,
- M-ACTION - Request an action to be performed on a managed object and
- M-EVENT-REPORT - Send events occurring on managed objects.

9.5.1.2 Discovery of Data Center Components

Discovery is the process of detecting and locating the presence of data center components and making them known to the management system.

One assumption of the standards and protocols discussed in section 9.5.1.1 is that the management system knows about the managed components and knows their addresses on the management network through which they can be reached. It has always been a problem to keep the inventory database used by the management system in a state that is consistent with the reality in the managed environment where inventory constantly changes due to scheduled replacements, new acquisitions of components or removal of defective components.

The *Service Location Protocol* (SLP) is a service discovery protocol that allows finding and locating computers and other devices a local area network environment without prior configuration. SLP has been designed to scale from small, unmanaged networks to large enterprise networks. It has been defined in RFC 2608 [RFC2608].

Discovery usually utilizes a multicast or broadcast issued to the LAN environment expecting that components which are currently available on the network respond with their location and other information.

9.5.1.3 Domain-specific Standards

Other standardization efforts comprised domain-specific such as the models and protocols defined by the Storage Networking Industry Association (SNIA) [SNIA] for the domain of storage.

9.5.2 WBEM and Web Services-based Management Protocols

Web-Based Enterprise Management (WBEM) [WBEM] is an initiative coupling CIM and Internet standard protocols and encodings (such as XML and HTTP) with the Common Information Model (CIM) [CIM]. The WBEM architecture includes the notion of a CIM server and various providers of management data, such as instrumentations. The CIM server acts as an information broker between the providers of instrumentation data and management clients and applications. This approach shields providers from management clients and applications.

WBEM consists of a set of management and Internet standard technologies standardized by the Distributed Management Task Force (DMTF) [DMTF] with the goal to unify the management of enterprise computing environments. WBEM provides the ability for the industry to deliver a well-integrated set of standard-based management tools leveraging the emerging Web technologies. The DMTF has developed a core set of standards that make up WBEM, which includes a data model, the Common Information Model (CIM)

standard; an encoding specification, xmlCIM Encoding Specification; and a transport mechanism, CIM Operations over HTTP.

In contrast to SNMP, managed objects do not only exist in managed components, they also exist in the CIMOM through which control is exercised by altering their states. State changes to managed objects in the CIMOM which have associations with managed components are reflected to those components via the WBEM protocols. In reverse direction, managed components can act as providers delivering data to managed objects in the CIMOM. Figure 82 illustrates two approaches of SNMP with CIM/WBEM-based management.

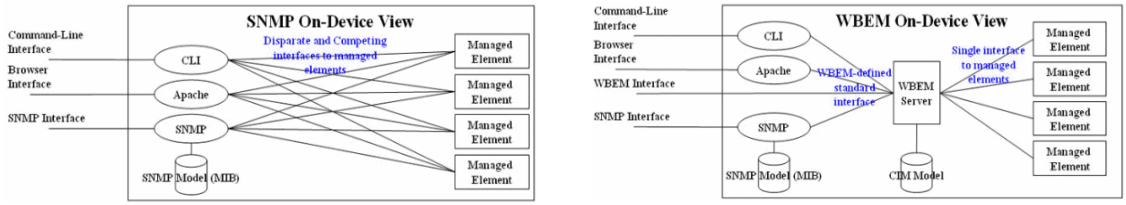


Figure 82: Comparison between SNMP and CIM/WBEM-based management.

Figure 83 shows a management environment built based on CIM data models and WBEM protocols with the information repository, the CIM Object Manager (CIMOM), in the center. Operators can interact with the CIMOM via consoles for inspection and manipulation. External programs can be connected to the CIMOM as well using the WBEM protocol. The WBEM Server acts as the application container serving the web services interactions.

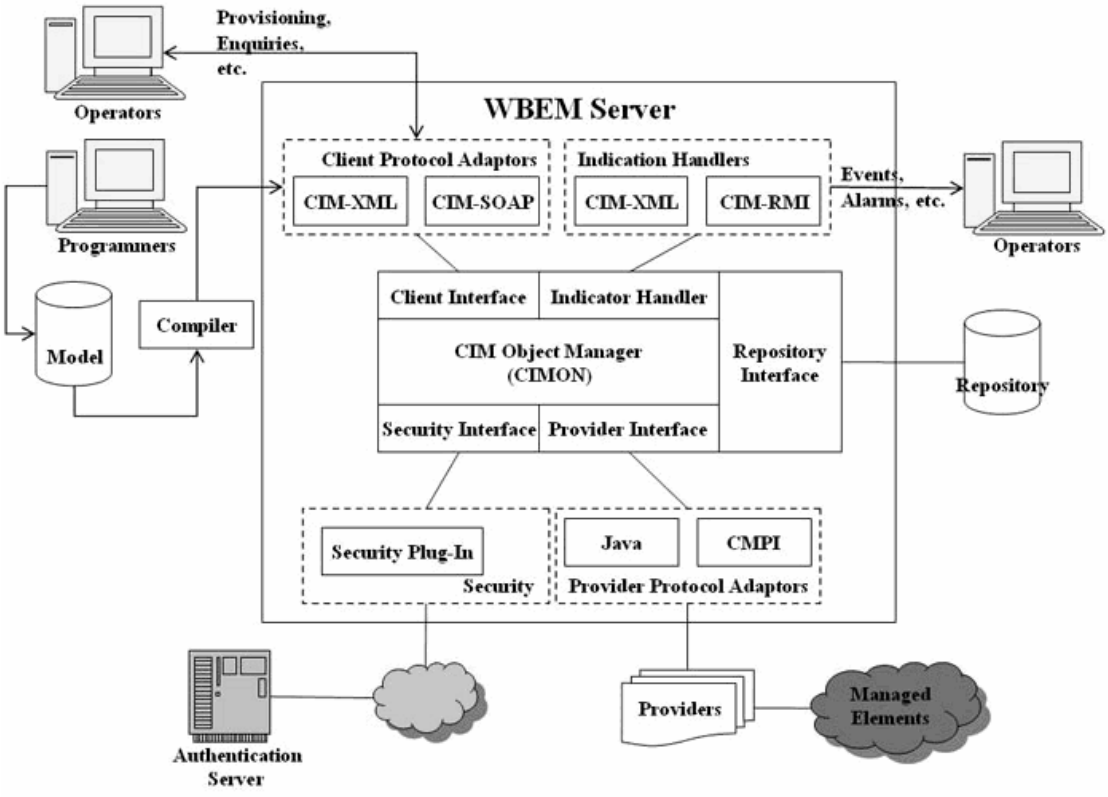


Figure 83: Management environment based on CIM and WBEM.

CIM uses HTTP/XML-based web services interactions with external systems. A number of protocols have been defined such as xmlCIM [xmlCIM], WSDM and WS-Management [WS-MAN]:

- *xmlCIM* is a standard way to represent CIM data using a Document Type Definition (DTD) to map CIM objects into XML elements [xmlCIM].
- *Web Services Distributed Management (WSDM)* [WS-DM] is a web service standard for managing and monitoring the status of other services using web services protocols which have been defined by OASIS [OASIS] in two specifications:
  - *Management Using Web Services (MUWS)* — WSDM MUWS defines how to represent and access the manageability interfaces of resources as Web services. It defines a basic set of manageability capabilities, such as resource identity, metrics, configuration, and relationships, which can be composed to express the capability of the management instrumentation. WSDM MUWS also provides a standard management event format to improve interoperability and correlation.
  - *Management Of Web Services (MOWS)* — WSDM MOWS defines how to manage Web services as resources and how to describe and access that manageability using MUWS. MOWS provides mechanisms and methodologies which enable manageable Web services applications to interoperate across enterprise and organizational boundaries.
- *WS-Management (WS-Man)* is another, related standard defining a SOAP-based protocol for the management of servers, devices, applications and more. WS-Management is developed by DMTF [DMTF]. The specification is based on open standards for Web Services and defined in [WS-MAN].

WBEM is comprised of the CIM model that we described in the previous section, the xmlCIM encoding of CIM elements in XML, and CIM over HTTP specification that enables interoperation across CIM systems.

Since WBEM defines the XML rendering of the CIM data structures and all of the above protocols are capable of transporting XML, WBEM does not depend on a specific transport protocol as long as it implements a number of web services methods that have been defined for WBEM and include operation types for data, metadata, queries and methods. The list of WBEM operations is shown in Table 15.

GetClass, EnumerateClasses, EnumerateClassNames, GetInstance, EnumerateInstanceNames, GetProperty, SetProperty, CreateInstance,	ModifyInstance, DeletInstance, CreateClass, ModifyClass, DeleteClass, Associators, AssoicatorNames,	References, ReferenceNames, ExecQuery, GetQualifier, SetQualifier, DeleteQualifier, EnumerateQualifiers.

Table 15: Basic WBEM operations.

DMTF has developed a core set of standards that comprise WBEM adding an encoding specification (the xmlCIM Encoding Specification) and a transport mechanism (CIM Operations over HTTP) to the Common Information Model. The xmlCIM Encoding Specification defines XML elements (described in a Document Type Definition, DTD or XML Schema) representing CIM classes and instances. The CIM Operations over HTTP Specification defines how the CIM classes and instances are created, deleted, enumerated, modified and queried. Also, the specification defines a notification and alerting mechanism for CIM events.

The following section introduces the implementation framework which had been chosen for actual realizations.

9.5.3 OGSI-based Implementation

The Open Grid Services Infrastructure (OGSI) [OGSI] was published by the Global Grid Forum (GGF) [GGF] as a proposed recommendation in June 2003. It was intended to provide an infrastructure layer for the Open Grid Services Architecture (OGSA) [OGSA].

OGSI today is obsolete. It was however used for a number of realizations for interacting with Data Center Components using the Globus Toolkit [GT4] middleware implementation which supported the Web Services Distributed Management (WSDM) [WS-DM] standard.

The OGSI-based implementation of web services-based management middleware is discussed by the realization of a Resource Flex Controller as part of the Resource Acquisition Manager presented in chapter 8.

This controller had the task to dynamically adjust server resources based on monitored load conditions. The process of adjusting the amount of resources according to demand is called flexing. The controller was implemented as a Management Web Service as part of the *Management Using Web Services (MUWS)* concept of WSDM. The controller interacted with so-called Control Plug-ins which acted as web-services counter parts in the managed environment observing actual load conditions on server resources.

The detailed description of this realization has been published in [Gra04d].

9.5.3.1 Component Design

The split of the control loop into a part that contains assessment and adjustment (Control Plug-in) and the resource infrastructure delivering monitoring events and providing the capability to actuate server group adjustments leads to the following design of components that are implemented as OGSI services:

- *Control Plug-in* – implements assessment and making adjustment decisions, contains control policy, and is customizable by application administrators or data center operators.
- *Interface Instances* – counterpart for one Control Plug-in in the resource infrastructure. It provides port types for actuating flex decisions and delivery of monitoring and state change events.
- *Interface Factory* – provides a port type for creating Interface Instances. A Control Plug-in initially contacts the Interface Factory in order to create an Interface Instance, which it then uses for all subsequent interactions.

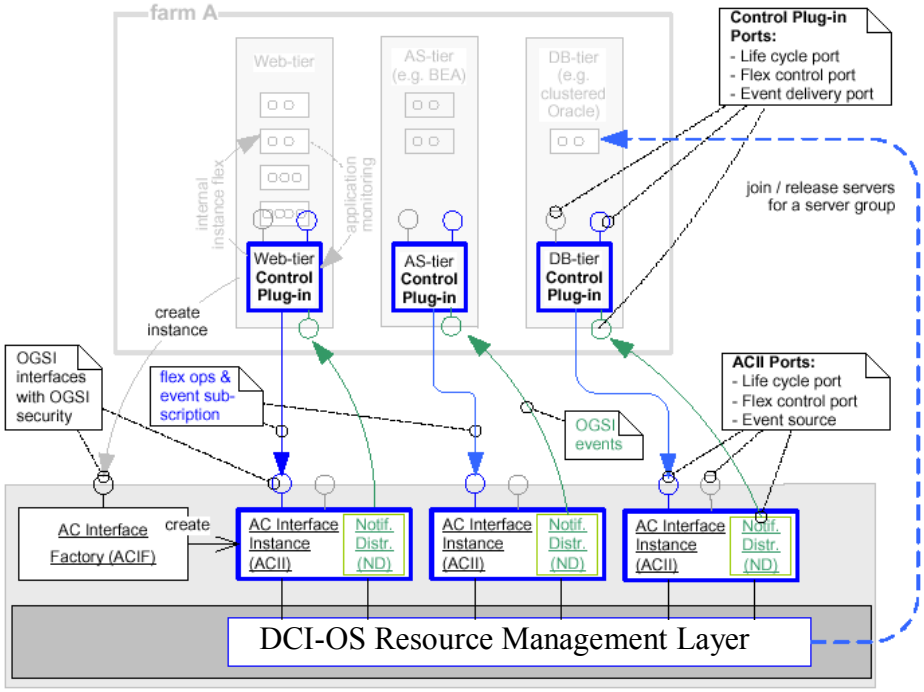


Figure 84: OGSI component design and basic control flow.

Figure 84 shows the OGSI components that are used to implement the control system. The figure shows a farm with three server groups (tiers) of web servers, application servers, and a clustered database in the upper part. Each server group has a Control Plug-in associated that is connected via OGSI to its Interface Instance counterpart in the resource infrastructure.

One Control Plug-in is connected to at most one Interface Instance. An Interface Instance receives flex operations from its associated Control Plug-in and translates and delegates them into corresponding operations in the resource infrastructure, and finally into the infrastructure controller software for actuation. Monitoring and state change events are delivered in the reverse direction from an Interface Instance to its associated Control Plug-in.

OGSI's event and life cycle mechanisms are used for OGSI components. Each has a number of port types:

Control Plug-in:

- Event Delivery Port – primary port for event delivery from the associated Interface Instance or other event sources such as application-level instrumentation.
- Flex Control Port – allows a higher-ordered control system to connect to and instruct a Control Plug-in.
- Life Cycle Port – for managing the life cycle of the control loop (start, stop, suspend, resume, etc.) and the Control Plug-in.

Interface Instance:

- Flex Control Port – primary port for receiving server flex operations from the associated Control Plug-in.

- Life Cycle Port – port for life cycle operations on Interface Instances.
- Event Distribution Port – port where Control Plug-ins can subscribe for monitoring and state change events.

Interface Factory:

- Factory Port – initial contact point for a Control Plug-in that allows the creation of one Interface Instance after its authenticity and authorization has been validated.

9.5.3.2 Crossing Protection Domains

Since the infrastructure controller software is performing critical resource management functions in the Utility Data Center, it is located in a protected domain that cannot be reached from applications outside.

Since Control Plug-ins are located outside the infrastructure controller software in order to meet the customization and integration requirements, the interaction between Control Plug-ins and Interface Instances must cross the protection domain boundary in a secure way (in analogy to system calls in operating systems).

OGSI offers a built-in security model that is used for securely crossing protection domains between the Control Plug-in residing outside and the Interface Instance residing inside the protected infrastructure controller software domain.

OGSI Security is applied at two stages:

1 Control Plug-ins are authenticated. A Control Plug-in's certificate is validated by the Interface Factory at initial contact. Only Control Plug-ins with registered certificates can create Interface Instances and are able to subsequently interact with the infrastructure controller software.

2 Establishing an encrypted communication channel between the Control Plug-in and the Interface Instance ensuring for both sides that the exchanged information is authentic and unaltered.

For the DCI-OS, the use of standards for information models and protocols has had a high priority. As particular standards had been investigated the widely established Common Information Model (CIM) as general Information Model of the DCI-OS and more recent web services-based protocols for management such as Web Services Distributed Management (WSDM) with an implementation using the Open Grid Services Infrastructure (OGSI) and the Globus Toolkit [GT4].

9.6 Summary

This chapter presented the DCI-OS layer, and with it, the actual "core" of the Data Center Infrastructure Operating System. The DCI-OS Layer is responsible to organize the resources found in a data center as pools from which quantities expressed as time profiles can be allocated to Infrastructure Services instances. Shortly before allocations become due, actual resource instances are assigned to allocations and provided to the requesting Infrastructure Service.

In order to provide this functionality, new abstractions had to be developed for the information model the DCI-OS operates upon. These extensions primarily relate to a substantially extended view of the abstraction of a *Resource*, one of the core concepts in the information model. This new abstraction included concepts of physical resources (the

devices as they exist in a data center), the concept of resource atoms (as smallest dividable resource units which can be created or obtained) and the principle of *Resource Construction* as fundamental concept. Traditional resource management systems struggle with the concept of a resource construction by only considering physical resources. The concept of a Resource Construction not only provides the means to express relationships between physical and virtual resources, which are omnipresent in today's data centers, it also allows expression and handling of complex resource compositions such as stacked resources including software resources comprised of machines, operating system and application software and considering it as one constructed resource.

Another important concept was the introduction of a concept of views on resources and *Ground Resources*. Ground resources were defined as resources for which pools (of resource atoms or resource construction) existed in the DCI-OS, but which could be exposed to requesting Infrastructure Services by different properties. For example, an Intel-based machine could be exposed as a capability to Infrastructure Services and could be grounded into a physical machine, a partition on a machine or a virtual machine, if no further properties would restrict these groundings. This allowed separating descriptions of requested resources from available resources which are changing with changing inventories. It also allowed flexibility for the DCI-OS in allowing different ways to construct resources, which is an essential property for the data center operator to utilize infrastructure efficiently.

Another major concept of the information model of the DCI-OS is the concept or a Resource Topology, which allowed to not only describe resource demands as resource sets, but in their full context of how resources relate to one another in the context of an Infrastructure Service, including connection and configuration properties required on resources when they were obtained from the DCI-OS. This information allowed the automation of finalizing deployment tasks on resources making them ready to run the applications in the Infrastructure Service. In today's practice, data center resources must be manually configured based on informal knowledge of domain experts in order to make them ready to run the target applications. Since configuration information is not formally represented in a model, deployment automation is hard to achieve. The concept of a Resource Topology provides this capability and hence enables this automation, which was a fundamental goal of the DCI-OS.

The ability to not only deal with current inventory in a data center, but also future (planned) inventory enables the DCI-OS to manage resource allocations as commitments of demanded resource capacity to Infrastructure Services. The process of resource allocation in data centers today is a human process and can be taken over by the DCI-OS. The concept of grounding and resource views can help to isolate demands from changes in the data center inventory.

The chapter presented the case for representing the core information in a commonly used and standardized framework (section 9.1) with the Common Information Model (CIM) as the chosen framework for the DCI-OS (section 9.2). Extensions for were presented for CIM reflecting the extended capabilities the DCI-OS needed (in section 9.3 – The DCI-OS Information Model).

Section 9.4 – The DCI-OS Resource Management Layer then presented the concepts of resource pool management with the extended capabilities of resource constructions,

resource topology allocation, resource grounding and resource demand and capacity profiles maintained by resource pool managers. Resource assignment was presented as a scheduling and optimizing task for which a substantial amount of research had been conducted and published.

Section 9.5 presented The Data Center Component Layer with resource pool drivers controlling the associated data center components (servers, storage, networking devices…). This layer corresponds to the layer of device drivers in an operating system with the difference that the control interfaces and protocols connecting the components with the control software executing on dedicated management machines relate to remote network protocols, for which a number of standards have been established by the industry over time with web services-based management middleware being the most recent one. A number of these technologies are discussed in this section.

The tasks of the DCI-OS have focused around resource management, which are an essential task in data center management, but are not the only category oft tasks. Following a traditional task categorization in IT Management in tasks related to Fault, Capacity, Availability, Performance and Security (FCAPS) management, mainly questions of Capacity, Availability and Performance have been addressed in the DCI-OS.

Fault and Security Management and others have not been addressed and are left to future extensions of the concept of a Data Center Infrastructure Operating System.

Chapter 10

Summary and Conclusions

The continued reliance on human labor for performing most management tasks, including lower-ordered routine tasks of IT infrastructure management, cannot meet the challenges data centers face today. Automation is a key approach for addressing the challenges of increasing scales of data centers, achieving higher agility of IT services to better support the business, and to reduce the overall operational cost of data center management. Automation of data center management is difficult and faces substantial obstacles. The data center automation market today remains fragmented focusing on specialized point solutions and technologies. No technical system has been demonstrated to date that transparently and automatically can manage the IT infrastructure of a data center.

This thesis focused on data center IT infrastructure management, which is the lowest IT infrastructure layer in a data center. Research conducted by the author at HP Labs from 2003 to 2008 exposed a number of deep technical problems, which have been addressed by research over the years. One of the insights was that many problems in data center automation are interrelated. Solving them in isolation does not produce the new quality of systems that is needed to manage data center infrastructures comprehensively.

Considering this insight and combining it with an earlier background the author had in operating systems [Gra97] led to the idea of developing a systematic approach to data center infrastructure automation by adopting concepts from operating systems and combining them with concepts from IT management. The consolidated result is the *Architecture for the Automated Management of Data Center IT Infrastructure*.

The disciplines of IT management and operating systems have largely been considered independently in the past. Combining both – while recognizing the differences – created new insights and provided the framework for defining the architecture of the DCI-OS, the Data Center Infrastructure Operating System. New insights and abstractions could be obtained from a new perspective that can help construct new automated data center management systems addressing some of the fundamental technical problems. This section summarizes these insights and the solutions that were discussed in this thesis.

First, this section revisits the questions that were asked in the introduction in Chapter 1:

1. What are the fundamental problems of data center IT infrastructure automation?
2. Can a comprehensive set of requirements be formulated for the automated management of data center IT infrastructure?

3. How can data center infrastructure automation systematically be achieved?
4. Which abstractions need to be developed upon which the architecture for the automated management of data center IT infrastructure can be built?
5. Which concepts and techniques from operating systems can be adopted for data center IT infrastructure automation?
6. Which concepts from IT management must be accommodated, which are not present in operating systems?
7. Which new abstractions and concepts must be developed that can neither be derived from operating systems nor from IT management?
8. How can the information model for these abstractions be defined?

Question 1 asked for fundamental problems of data center infrastructure automation. The discussion in this thesis has exposed a number of reasons:

- Management is fragmented, technically (with a diversity of specialized management systems) as well as organizationally (with the diversity of specialized groups of people performing management tasks). Most management systems assist people in specific tasks, but do not perform management tasks autonomously.
- Management fragmentation leads to information fragmentation across systems and people leading to information duplication, redundancy and inconsistency. It is unfeasible to build automation (algorithms, policies, decisions) based on inconsistent information.
- Information fragmentation and the lack of common abstractions are also the cause of lacking integration of the various management systems that coexist in a data center.
- Substantial information is carried by people outside of management systems. Planned, designed or expected states are often not represented in information models in management systems. The lack of this information prevents systems from automating processes towards achieving those states such as for resource allocation, deployment or operational management.
- Relying on discovery as primary source of management information is insufficient since it only captures physical components that currently exist in the data center and only their current state.
- Technologies are being introduced in data centers today that have no appropriate expression in information models in management systems, such as:
  - the recognition of constructed resources beyond physical components, including their construction and mapping relationships;
  - the decoupling of applications and data from physical components;
  - the shared use of resources;
  - the dynamic provisioning of resources based on actual demand;
  - the recognition and representation of interconnected lifecycles of services, software, data, systems and components in data centers.
- The lack of appropriate abstractions leads to a lack of concepts upon which a comprehensive automation architecture can be defined.

The combination of these problems has prevented the integration of management systems and the emergence of comprehensive automation systems in data centers to this date although attempts have been made. The attempt of an early data center automation solution called the HP Utility Data Center (UDC) was analyzed in section 4.1. UDC's failure in fact initiated the deeper-reaching research in HP Labs this thesis is based on. The analysis of UDC's shortcomings led to the detailed requirement specification in Chapter 4 answering *question 2,* which had asked for a set of comprehensive requirements for an automated data center IT infrastructure management system.

Question 3 asked how a systematic approach to data center infrastructure automation can be achieved. A first and fundamental observation was that the problems of data center automation are interconnected. For example, because people manage IT through direct manual intervention in the managed environment, the information models in management systems become inconsistent and continuously need to be rediscovered for update. The principle of discovering changes made in the managed environment after the fact is fundamentally flawed. It leads to continuous windows of inconsistency between the state of managed elements and their representations in the management information model. This inherent inconsistency in turn poses a significant obstacle to automation that is relying on consistent information. The conclusion from these interdependencies is that they must be solved in combination breaking the dependencies. A solution to the specific problem of inconsistent management information models has been developed with the IT management automation controller that was presented in section 8.1 and that is summarized in section 10.3.

Solving problems in combination means that they must be addressed systematically. The architecture of the DCI-OS that was presented in Chapter 5 summarizes the functional, structural and organizational aspects of a comprehensive approach to data center IT infrastructure automation. It also represents the answer to the *hypothesis* of this thesis, which has been to demonstrate the systematic development of the architecture for the automated IT infrastructure management in a data center that is rooted in concepts of operating systems and IT management.

Question 4 asked for a set of abstractions that needed to be developed upon which the architecture of a DCI-OS can be built. The following sections 10.1 (concepts from operating systems), 10.2 (concepts from IT management) and 10.3 (new concepts) summarize these abstractions that were presented in this thesis and their adaptation for the DCI-OS.

10.1 Summarizing Concepts Adopted From Operating Systems

Question 5 asked which concepts and techniques from operating systems can be adopted for data center IT infrastructure automation.

Figure 1 in Chapter 2 presented a categorization of operating system concepts. The following summary highlights some of these concepts that have been adopted for the architecture of the DCI-OS.

Layers. The concept of layers is fundamental in operating systems. Layers introduce structure, scope and containment. The concept of layers allows separating concerns. It allows the encapsulation of domains in which abstractions (elements with certain qualities) exist. A layer of one degree builds upon a layer of a lower degree. Abstractions

that exist in one layer are created by the underlying layer. The architecture of the DCI-OS employs three main layers to separate three main domains. Layer 1 is the planning and design layer, layer 2 is the infrastructure services layer, and layer 3 is the DCI-OS layer with various sub-layers for the information model, resource management, resource pools, resource pool drivers, and for data center components.

The concept of a **Hardware Abstraction Layer** (HAL) was adopted from operating systems as the interface layer between the layer of resource pools and the physical device driver layer introducing normalized, standard web-services-based interfaces, protocols and data model representations. These layers exist on the data center side of the DCI-OS architecture (the data center-related part in Figure 7). Two layers were introduced for the **run-time system** and for dynamic resource acquisition, deployment and task automation on the application side (the infrastructure services-related part).

Resources. Another key abstraction adopted from operating systems is that of a resource. The architecture of a DCI-OS employs the concept of a resource as the unified abstraction of an entity providing computational services that are needed and used by other computational services of the same or of a higher degree. The general resource abstraction unifies the three dimensions of computation, storage and communication, which are still managed independently in data centers today.

Defining resources relatively (one needed by another) allows to establish (and later to represent in information models and to construct and manage in actual systems) complex resource construction and mapping relationships among resources. Although the concept of viewing resources relatively to one another is present in operating systems (e.g. the file system abstraction mapped onto a set of block storage resources), it was broadened for the DCI-OS to not only reflect mapping relationships of resources from one degree into a lower degree, but also to capture relationships among resources of the same degree introducing a new construct called a *Resource Topology*. A resource topology defines the set of resources that is needed for an entire application execution environment comprised of server, storage, and network resources. A resource topology represents not only the resource set. It also includes the relationships among the resources and the configuration information for constructing each resource and the linkages between them for supporting the desired application execution environment. Capturing this detailed information later enables the automation of configuration and deployment processes that are constructing the resources as they are needed for the application execution environment.

Application is a concept in operating systems representing application code, data and processes that are needed for performing useful computations. The concept of an application was adopted for the DCI-OS in form of a set of application services executing in the application execution environment that is produced and managed by the DCI-OS.

Application execution environment is a concept in operating systems representing the layer in which all resources exist in the form as they are needed by the applications. Resources are created and supplied by the underlying operating system. This concept was adopted in the architecture of the DCI-OS in form of the application services execution environment that is comprised of infrastructure services provided by the resources that are defined by a resource topology. The application execution environment is the essential abstraction created by the DCI-OS allowing application services to execute.

Run-time control is a component of the operating system that resides inside the application environment allowing it to interact with the operating system. This concept was adopted in form of the run-time driver layer in the architecture through which resources can be acquired and released as well as lifecycle operations can be issued on explicit request.

Resource pools describe the abstraction of a container in which an operating system maintains unassigned resources of the same type as fully constructed and ready to use entities. Resource pools serve as buffers to quickly supply resources when they are needed. They also allow reusing resources when they are returned without destructing them. Resource pools provide reservoirs from which unused resources can be quickly acquired and released. Pool-oriented management of ready-to-use resources has recently become an accepted practice in data center resource management.

Resource sharing and **dynamic resource management** had been early concepts in operating systems which just recently have been introduced in data centers for the same reason: to increase resource utilization and to make more economic use of resources in data centers. Concepts of sharing and dynamic resource provisioning have been incorporated into the architecture of the DCI-OS as core properties that are supported by managing dynamic construction and mapping relationships. It includes mechanisms for automated scheduling and decision making using closed loop controllers (see the adaptive flex controllers in section 6.5).

Scheduling is a concept from operating systems that is related to resource sharing and dynamic provisioning. Scheduling describes the process of defining an order (a schedule) of assigning a set of resource requests onto a set of available resources. One form of scheduling is pre-establishing the allocation plan as a planning task. Another form is making scheduling decisions dynamically during operation involving automated decision making. The decision-making as part of dynamic scheduling includes policy for guiding automated decisions. The execution part assumes a mechanism for performing mapping operations of requested resources on a shared resource set. This capability is supported by the DCI-OS by managing dynamic resource constructions and mapping relationships.

In data centers, scheduling traditionally has been a planning activity performed by people determining static resource allocations for applications. With the introduction of dynamic resource provisioning techniques, scheduling extends from a planning task performed by people to a continuous run-time activity requiring automation for monitoring, decision-making and resource adjustment processes. Controller concepts have been successfully demonstrated as automation patterns for that [Zhu08], [Gma10].

While scheduling in operating systems is a multiplexing technique of current resource demand onto currently available resource inventory, scheduling in a data center must take the overall lifecycles of services, applications and resources into account. For this reason, the concept of scheduling had to be significantly broadened into separate phases of resource allocation (in terms of generally required resource capacity) and resource assignment (the selection of specific resource instances providing that capacity at a specific time). Choices needed to be evaluated for selecting and constructing the resources for assignment. During run-time, resources could be flexed dynamically depending on actual demand.

Process management in operating systems is often referred to as the multiplexing technique of concurrently executing a number of application processes on fewer processors. But operating systems also execute management processes such as creating new process instances and loading applications into processes. These management control processes have been adopted in a generalized form of lifecycle processes for operational management tasks in the DCI-OS. Examples are the automated resource request workflows in section 9.4.2, the automated deployment workflows in section 8.2 or the processes performed by the resource flex controller in section 8.3.

Isolation and **protection** refer to concepts in operating systems to avoid interferences of applications among each other when executing in a shared environment. These concepts were factored into the DCI-OS by including the appropriate configurations in resource topologies for servers, networks and storage implementing the desired isolation policies.

10.2 Summarizing Concepts Adopted From IT Management

Question 6 has asked for concepts from IT management that need to be accommodated by a DCI-OS, which are not present in operating systems.

Figure 2 in Chapter 3 presented a categorization of IT management concepts partitioning the environment into a **managed** and a **management environment**. The concept of a managed and a management environment were adopted by partitioning the layers of the DCI-OS architecture into managed and management sections.

IT Service is a central concept in **IT Service Management** [itSMF]. An IT service represents an abstraction of the user's perspective of a set of functions IT delivers. It is a central concept in ITIL [ITIL]. Since the DCI-OS focuses on data center IT infrastructure, the concept of an IT services has been adapted to infrastructure services that are forming the overall execution environment for application services. It is a central abstraction of the infrastructure services layer in the DCI-OS architecture.

Lifecycle describes an abstraction covering the various stages and transitions each managed and management element passes through. Capturing pre-existence lifecycle stages is essential for representing future states, plans and designs. The concept of lifecycle was adopted by expanding the technical architecture of a DCI-OS with a distinct planning and design layer and by including workflows for resource lifecycles in the DCI-OS for allocation, assignment, deployment and the operation for infrastructure services.

Management information model describes a concept in IT management in which all information that is relevant for IT management is represented in a unified form and in a state that is consistent with the reality of the managed environment. The information model for IT management defines the core data structure upon which systems and algorithms operate, make decisions and perform automated processes. It is consequently a core part of the DCI-OS architecture and is represented as a distinct information model layer. The DCI-OS information model layer is separated into a data center-related part, covering the information about the data center, and into an infrastructure service-related part, covering the information related to the application services environment. Separating the information model into these two parts is a significant extension over existing management information models that do not make this distinction. The separation was introduced to decouple the development of application services and their supporting resource topologies independently from specific data center conditions (inventory,

availability, specific data center constraints) supporting *reuse* and transferability of application services and resource topology designs from one data center into another.

The problem of lacking uniformity in management information models was addressed by choosing the Common Information Model (CIM) [CIM] as representation framework for the DCI-OS information model. CIM is an established information model standard defined by the DMTF [DMTF]. Extensions were introduced to the CIM core models in order to represent the set of abstractions needed for the DCI-OS. A number of new first-class entities were introduced to represent *actors*, *roles*, *activities*, *relationships*, *contexts*, *views*, *policies* and *resources* (see section 9.3, The DCI-OS Information Model). Those concepts were needed to not only represent the elements of the managed environment, but also the context in which automated management tasks occur. Capturing contextual information beyond managed elements in management information models is another extension introduced in the information model of the DCI-OS.

Further data model representations have been developed for the new DCI-OS abstractions, which are summarized in section 10.3, such as for *resource topologies*, *resource constructions*, *complex resource request* and *allocation formats*, *resource capacity* and *demand profiles*, *desired states* and *observed states*.

Management integration is the concept (and practice) to enable management systems to not only coexist in a data center, but also to work together for assuming more complex management tasks. Management integration requires integration of information models. CIM was chosen for the DCI-OS as normative information representation. CIM's provider concept allows encapsulating proprietary managed and management systems and normalizing their external interactions using the CIM data representation. The provider concept consequently was adopted by the DCI-OS as a general pattern for integrating proprietary management capabilities. Sections 8.2 demonstrated the integration of a HP proprietary deployment system into the DCI-OS context [Gra05a].

Unified management protocols are a related topic to management integration aiming at standardizing and unifying the interactions with proprietary managed or management components. As part of that research, a number of web services-based management standards and middleware platforms were investigated such as OGSI, WSRF and WSDM. [Gra06] discusses requirements for management middleware. [Gra06a] describes a middleware platform for delivering IT management services that was developed during that time. In particular, WBEM and OGSI-protocols were used for implementing the IT Management Automation Controller [Gra04c,d,e] (presented in section 8.1) and for the Flex Automation Controller (presented in section 8.3). The work on web-services management standards for the integration of management systems has been summarized in a book: “Web Services in the Enterprise: Concepts, Standards and Management” that was co-written by the author and published by Springer in 2004 [Gra04b].

Model-driven management. The problem of maintaining consistency between the states in the managed environment and their representations in the information model in the management environment was addressed in a new way. Typically, changes are made in the managed environment first, which are then periodically rediscovered updating the management information model afterwards. This principle is also referred to as *model-based management*. It has been built into most IT management systems today. The fundamental problem with model-based management is its inherent inconsistency with

the reality of the managed environment that is caused by allowing changes to be made in the managed environment without updating the model.

Reversing that order is key to addressing the inconsistency problem. It means that intended changes need to be made in the information model first, e.g. by setting the flag through a console indicating the bootstrap of a server, for example. This change in the information model is then programmatically actuated in the managed environment by sending the corresponding signal to the server. This principle is referred to as *model-driven management*. It was adopted for the DCI-OS as a fundamental principle. [Gra05b] demonstrated how model-driven management can be integrated into a conventional software configuration system from HP. Discovery is still needed in order to report when the intended state change has actually occurred in the managed environment and to detect problems or error. A major innovation developed by the author was the extension of model-driven management into a closed-loop, discrete state controller pattern for IT management automation in section 8.1. It combines the control flow of model-driven change actuation with the discovery control flow that is detecting when changes have been actuated or abnormal conditions have been observed.

Best-practices from frameworks such as ITIL were adopted for the architecture of the DCI-OS where those practices could be cast into technical realizations or implemented as integrated tool chains. It mainly occurred in the planning and design layer (Chapter 7). This layer provides tools for capacity planning (Section 7.2), inventory management (as part of the DCI-OS information model), application and infrastructure sizing (such as for the performance engineering process presented in section 7.3) and change management as part of deployment automation (section 8.2).

10.3 Summarizing New Concepts and Significant Extensions

Question 7 has asked for new abstractions and concepts which had to be developed for the architecture of the DCI-OS which could neither be derived from operating systems nor from IT management. Most of the concepts discussed in the following were developed by the author and first documented in an early architectural description [Gra03b] that represented a comprehensive design of a data center automation system. The concepts evolved over time and were refined later over the course of research.

Infrastructure Service is an adaptation of the concept of an IT service. An infrastructure service provides a fully operational execution environment for an application service. It comprises a set of configured and operational computing, networking and storage components that are required by infrastructure services as a whole for executing application services. An infrastructure service is the unit of allocation, assignment, deployment and operation in the DCI-OS. It also is the unit of isolation and protection.

Resource Topology describes the design model of an infrastructure service. A resource topology defines the entire environment of resources including their configurations which, as a whole, comprises the resources for an infrastructure service. A resource topology specifies the types and quantities of resources in combination with their configurations that are needed for constructing and managing the resulting infrastructure service. During deployment, information from the resource topology model is applied to assigned resources from the data center environment.

Resource topology (the design model) and infrastructure service (the manifestation of a resource topology at run-time) are central abstractions of the DCI-OS architecture. They are new abstractions that do not exist in operating systems or in IT management.

Resource construction is a concept which partially exists in operating systems by recognizing the fact that resources do not only exist as physical components in machines or data centers, but must be considered relatively building one upon another. Constructed (aggregated, transformed, virtualized) resources must be recognized as fully qualified and identifiable elements in the system. Operating systems only construct limited sets of resources such as processes on processors or virtual memory on physical memory and disk space. Complex constructions such as resource topologies are not known in operating systems.

Resource Topology design describes the process of creating the models for resource topologies. Design processes are not common in IT or in IT management where systems are usually planned, architected and deployed, lacking an explicit design step and lacking explicit formal design representations. IT architecture typically refers to defining general building blocks for IT components or systems. The further breakdown into designs and further into detailed configurations on systems usually occurs during deployment where design choices are made that are often not documented. The lack of detailed, formalized deployment specifications (designs) prevents the automation of deployment processes. It also leads to the lack of information about intended or desired state information, about what "should be" in the managed environment, which is essential information for the comparison to what "is" in the managed environment. Knowing both, what "is" and what "should be" is a precondition for automatically determining a valid state and automating operational management tasks as it is performed by the generalized IT management automation controller.

The existence of formalized, detailed models or designs of IT components or systems is a prerequisite for most automation capabilities. Other industries, such as the semiconductor industry, have developed sophisticated modeling techniques and tool chains that are going even further validating designs before their implementation and production. All these capabilities are enabled once modeling methodologies and tool chains are established. IT and IT management are just at the beginning of this development that is fostered by increasing automation.

Developing modeling methodologies and tool chains for IT designs is a major challenge. Figure 22 showed an early prototype of a *Resource Topology Designer* tool developed by the author in 2003, which generated a machine-readable model representation from a visually constructed resource topology. The resulting models could then drive further automation processes. Section 7.4 presented later research that demonstrated how design processes from different layers could be linked to systematically derive more comprehensive sets of IT configurations through several transformation stages supplementing the logical designs with non-functional requirements all the way to a fully resolved, deployable resource topology model. This specific research occurred in collaboration with SAP Research [Bel07], [Bel07a], [Gra08]. It resulted in another design tool developed by the author for SAP deployments called the *ModelWeaver* [Gra08a] and was targeted to HP's automated blade infrastructure software.

Introducing a design discipline in IT in combination with the creation of design tools and subsequent automation systems marks a major shift in how IT infrastructure, IT systems and IT services are designed, created, deployed and managed in future. The availability of detailed, machine-readable models will further foster automated management.

Policy-based design and configuration has been another research work the author has contributed to. The goal was to automate the construction of complex and detailed resource topology specifications from higher-level definitions using constraint-satisfaction techniques. Section 7.6 introduced this work. An extension was developed for the Resource Topology Designer tool that provided this capability. A number of publications document this work [Sah04], [Sah04a], [Gra05].

Grounding of resource specifications. The term grounding originated in Grid computing [Fost03] from the need to manage resource reservations more independently from specific resource properties that were available at a time in a data center. Those could change over time causing matching problems with existing reservations. Grounding is the process of mapping higher-order resource requirement definitions into concrete descriptions of resources in terms of specific types and quantities that are available in a data center at the time the reservation becomes due. For example, rather than requesting server resources with specific properties of CPU-type, clock speed, cache and memory sizes. Server resource can also be requested in higher terms of a certain processor architecture and a capacity metric expressed by some benchmark numbers. This definition than can be matched by a variety of concrete server types that are available in the data center. Grounding means the mapping of those higher-ordered resource definitions into concrete realization choices. Choices for these mappings increase the flexibility in the resource management system to optimize resource use. For instance, if servers of lower capacity are requested, the choice may exist to render them as a virtual machine rather than a dedicated physical server. The policy-based design approach has been applied to implement grounding using the same constraint satisfaction technique that was used for policy-based design and configuration [Sah04], [Sah04a], [Gra05].

To some extend, the practice of sizing in enterprise IT systems can be compared to the process of grounding. In sizing, capacity requirements are factored into the functional architecture of applications and systems. Decisions are made for concrete resources and systems in terms of types and the needed capacities. While sizing requires an expert today, constraint-based sizing has been automated and applied to cases of SAP application and system sizing as part of that research [Gra08], [Rol08].

Resource allocation traditionally has been a manual planning task in data centers. Grid clusters and high-performance compute farms have developed reservation systems that automate the management of allocation schedules. They are often restricted to managing compute resources, but do not take storage and networking resources into account. In the DCI-OS, resource topologies are the unit of allocation, deployment and operation, which include compute, storage and networking resources. A process of resource allocation had to be developed for the DCI-OS, which was described in Section 9.4.3. A *complex resource request format* with *capacity* and *demand profiles* is the central abstraction for that. The data representation and the match-making algorithms were developed during a Master's Thesis in HP Labs [Kön04].

Late binding of resource allocations to specific resource instances is another concept that had emerged from the work on resource allocation for resource topologies recognizing the fact that most applications today support a range of hardware and software platforms for deployment (within specified constraints) providing the DCI-OS with flexibility to construct those configurations that are most economic.

Resource Assignment is the last stage of pre-deployment resource management. It was presented in section 9.4.4. Research contributions mainly focused on optimizing resource selection from resource pools for constructing resource topologies.

The resulting effects of these new principles of policy-based configuration, grounding, the separation of resource allocation, and the late binding to resource instances are:

- *tolerance of change* in physical inventory in the data center;
- *reusability* of resource topology designs for different configurations and in different data centers;
- increased *efficiency* by utilizing configuration choices by the DCI-OS;
- the *automated generation* of deployment configurations enabling easier re-configuration by re-computing new deployment configurations using new sets of constraints;
- the *optimization* of resource assignments for making placement decisions.

IT Management automation controller. A fundamental shortcoming in model-based IT management systems has been that they only capture the currently observed state of managed elements. Based on only that, the management system cannot determine whether this state is the *intended* (or desired) state of the element or not. The management system can only report the current state to a human who possesses the information about the intended state, but cannot take action itself.

To enable automation, the concept of model-driven management was extended into the concept of an IT management automation controller that was presented in section 8.1. It maintains distinct representations of the *desired state* and the *observed state* about a managed element. The overall goal of the controller is to maintain alignment between the desired state model and the observed state model. The controller has two simultaneous control flows. One flow supports model-driven management. Intended changes are made to the desired state model first. They create a difference to the observed state model from which the controller derives a sequence of actions and applies them to the managed element facilitating the change. The other flow is the continuous discovery cycle that is updating the observed state model with the states detected from the managed element.

Both control flows work together in two ways. First, changes made to the desired state model are actuated by the controller in the managed environment based on the difference they create between the new desired state and the currently observed state. Once the change is effective in the managed environment, it is reflected back to the observed state model by the discovery cycle re-aligning both models again. This flow is called the affect loop. Second, in case unintended changes occur to managed elements, such as failures, the discovery cycle reports the observed change back to the observed state model, also causing a difference to the desired state model, based on which the controller can determine corrective actions and actuate them on the managed element. This is the correction loop.

While controllers have been used in IT management for automatically aligning numeric properties such as controlling server capacity to align with actually observed server demand, generalizing the controller concept to finite sets of discrete management states is another contribution by the author. Section 8.1 presented a realization of the IT management automation controller using executable Petri nets [Gra07]. An execution engine was built for this purpose. The controller concept was implemented and tested in a number of pilot engagements that were discussed in section 6.5.

The discrete state controller can be used as a general automation pattern for a wide range of automation tasks in data centers.

Finalizing the discussion of the questions that have been asked in the introduction of this thesis, *question 8* asked how an information model for the new abstractions of the architecture of the DCI-OS can be defined. The information model for the DCI-OS was presented in Chapter 9. Data model representations have been developed as extensions to the CIM core model for actors, roles, activities, relationships, contexts, views, policies and resources (see section 9.3) as well as for resource topologies, resource constructions with construction and mapping relationships, complex resource request and allocation formats, resource capacity and demand profiles, lifecycle states, and for desired and observed state models of managed elements that are needed for the IT management automation controller.

10.4 Open Issues and Conclusions

Data center automation is an ongoing effort requiring continued research and development. Confidence needs to evolve over time and over generations of automation systems. However, a broad implementation of this architecture has not occurred. It requires substantial effort to develop, test and deploy a data center automation system of this nature.

Vendors of automation systems have acknowledged the complexity of data center automation by scaling down the scope of automation solutions and systems. For example, HP's early automation technology (UDC, 2002) was still scoped for an entire data center. Current solutions focus on smaller domains, such as the domain of a rack or an isle in a data center. Integrated automation solutions are being built for these domains and are successful in the market. HP's Server Automation Software [HPSAS10] is a recent (2010) offering that provides rack-scale automation capabilities for farms of standard server and storage blades.

Another major issue and obstacle to data center IT infrastructure automation remains the complexity of the data models needed to drive complex automation tasks. Neither experience nor tools exist for creating and managing this complex information. This issue needs to be addressed by developing a design discipline with design tools for IT systems, solutions and IT infrastructures. Over the years, the author has developed a number of such design tools as prototypes addressing this issue. Technologies were developed to generate the detailed and complex information models for resource topologies from higher-ordered requirement specifications. More work needs to be done, not only for designing IT infrastructure in data centers, but also for designing systems and solutions in IT in general.

Other industries have successfully demonstrated the power and capabilities emerging from using computer models. Computerization of designs and design processes has allowed addressing scales that are beyond human capabilities, such as the millions of elements that are part of a modern semiconductor design. Scales in data centers have not reached into those dimensions, but they are growing demanding more systematic approaches and the development of design models and tools changing how infrastructures, systems and solutions in IT are being planned, designed, produced and managed. The development of a design discipline for IT is a prerequisite for this.

And finally, the fragmentation of management processes across systems, people and organizations remains an open issue that cannot easily be overcome. It is deeply built into IT organizations and the ways they operate.

Nevertheless, the efficient management of data centers is critical in today's information-centered world. Automation has proven to increase efficiency in other industries. Data center management, as part of overall IT management, must face the challenges of developing suitable and effective automation solutions in order to address the increasing scales of data centers, the desire for more agile IT environments, and to reduce operational cost. In conclusion, enterprise data center and IT management architectures need to accelerate progress in automation for achieving these goals.

10.5 Final Remarks

The established enterprise data center and IT architectures are increasingly being challenged by a new type of IT infrastructures that have evolved with the trend of "cloud computing". Those IT infrastructures are of extreme scales and are designed and built for scale from the beginning. Due to their scale, they are highly uniform and are highly automated. The automation they incorporate delivers unprecedented low operational cost per unit for IT infrastructure, systems and services.

Luiz Barroso and Urs Hölzle (from Google) describe these new data center infrastructures in their recent book *"The Datacenter as a Computer"* [Hölz09], for which a new class of management automation software has been developed from the ground up. Those infrastructures are increasingly being developed by the large Internet service providers. The subtitle of the book, *"An Introduction to the Design of Warehouse-Scale Machines"*, draws on the analogy to large-scale machines. It also implies the presence of automated management systems, or operating systems, that have been developed for those warehouse-scale machines.

This thesis considered the conventional domain of enterprise data centers and the problems and challenges for increasing automation in them. There is still a large legacy of applications in enterprises that cannot be migrated to the new "warehouse-scale machine" and hence providing the potential for significant innovation in this area.

However, there is no fundamental technical reason why the information processing functions enterprises rely on could not be delivered from these new, highly efficient, warehouse-scale machine infrastructures. A new generation of enterprise software and IT service delivery methods may emerge that are making this transition occur.

The *Architecture for the Automated Management of Data Center IT Infrastructure* that was presented in this thesis can serve as a blue print or as a general pattern for building more comprehensive automation solutions for enterprise data centers. It exposed a

number of fundamental problems and presented solutions that have been validated though experimentation and practical realizations. The goal of the fully automated, autonomous and transparent operation of data centers, however, remains a challenge requiring continued effort in research and development.

References

[Aal99] Aalst, Wil M. P.: *Formalization and Verification of Event-driven Process Chains*. Information & Software Technology 41(10): 639-650 (1999).

[Acc86] Accetta, M., Baron, R., Bolosky, W., Golub, D., Rashid, R, Tevanajan, A., Young, M.: *Mach: A New Kernel Foundation for UNIX Development*, Usenix Summer Conference, Atlanta, GA, June 1986.

[Andr02] Andrzejak, A., Graupner, S., Kotov, V., Trinks, H.: *Algorithms for Self-Organization and Adaptive Service Placement in Dynamic Distributed Systems*, HP Labs Technical Report, HPL-2002-259, September 2002.

[Andr06] Andrzejak, A., Graupner, S., Plantikov, S.: *Predicting Resource Demand in Dynamic Utility Computing Environments*, IEEE International Conf. on Autonomic and Autonomous Systems (ICAS 2006), Santa Clara, CA, July 19-21, 2006.

[ASN1] International Telecommunications Union (ITU) and OSI: *Abstract Syntax Notation One (ASN.1): Specification of basic notation*, http://www.itu.int/ITU-T/studygroups/com17/languages.

[Ass93] Assenmacher, H., Breitbach, T., Buhler, P. Huebsch, V., Schwarz, R.: *The PANDA System Architecture – A Pico-Kernel Approach*, 3rd Workshop of Future Trends in Distributed Systems, 1993.

[Ast94] Astrom, K.J., Wittenmark, B.: *Adaptive Control*, (2nd Edition), Prentice Hall, 1994.

[Bals04] S. Balsamo, A. Di Marco, P. Inverardi, and M. Simeoni, *Model-based Performance Prediction in Software Development: A survey*, IEEE Transactions on Software Engineering, vol. 30, no. 5, pp, 295-310, May 2004.

[Bard78] Y. Bard and M. Schatzoff, *Statistical Methods in Computer Performance Analysis, Current Trends in Programming Methodology*, Vol. III: Software Modeling, K.M. Chandy and R.T. Yeh, Eds. Englewood Cliffs, NJ:Prentice-Hall, 1978.

[Bart04] Bartolini, C., Boulmakou, A., Christodouku, A., Farrell, A., Salle, M., Trastour, D.: *Management by Contract: IT Management driven by Business Objectives*, 11th Workshop of the HP OpenView University Association (HPOVUA 2004), Paris, France, June 2004.

[Bart04a] Claudio Bartolini, Mathias Sallé: *Business Driven Prioritization of Service Incidents*. DSOM 2004: 64-75.

[Bel07] G. Belrose, K. Brand, N. Edwards, S. Graupner, J. Rolia, and L. Wilcock, *Business-driven IT for SAP - The model information flow*, Second IEEE/IFIP International Workshop on Business-driven IT Management (BDIM 2007) in conjunction with IM 2007, Munich, Germany, pp. 45-54, May 21, 2007.

[Bel07a] Belrose, G., Brand, K., Edwards, N., Graupner, S., Rolia, J., Wilcock, L.: *Adaptive Infrastructure Meets Adaptive Applications*, paper at HP TechCon 2007, San Antonio, Texas, April 22-25, 2007.

[Ber95] Berhad, B.N., Savage, S., Pardyak, P., Sirer, E.G., Fiuczynski, M., Becker D., Chambers, C., Eggers, S.: *Extensibility, Safety and Performance in the SPIN Operating System*, Proceedings of the 15th SOSP, pages 267-284, December 1995.

[Bla98] Blanding, S.: *Handbook of Data Center Management*, Auerbach publications, 2nd Edition, 672 pages, October 1998.

[Blan99] Blanding, S.: *Enterprise Operations Management Handbook*, Second Edition, 672 p., ISBN 0-8493-9824-X, Auerbach Pub, October 1999.

[Bom92] Bomberger, A.C., et.al.: *The KeyKOS Nanokernel Architecture*, Usenix Workshop on Micro-Kernels and Other Kernel Architectures, April 1992.

[Bon02] van Bon, J. (ed.): *IT Service Management: An Introduction*, van Haren Publishing, ISBN 90-806713-4-7, 2002.

[Boo98] Booch, G., Rumbaugh, J., Jaconson, I.: *The Unified Modeling Language (UML)*, Addison Wesley, ISBN 0-201-57168-4, 1998.

[BPF] TM Forum (TMF), *eTOM Business Process Framework,* http://www.tmforum.org/BestPracticesStandards/BusinessProcessFramework/6637/Home.html.

[Burg93] Burgess, M.: *Cfengine: A System Configuration Engine*, University of Oslo report 1993, http://www.cfengine.org.

[Buz73] J.P. Buzen, *Computation Algorithms for Closed Queuing Networks with Exponential Servers*, Communications of the ACM, vol. 16, no. 9, pp. 527-531, September 1973.

[Camp93] Campbell, R.H., Islam, N., Raila, D, Madany, P: *Designing and Implementing Choices: An Object-Oriented Operating System in C++*, Communications of the ACM, 36(9):117-126, September 1993.

[Can07] Cannon, D., Wheeldon, D.: *Service Operation*, Vol 4, ITIL Version 3, 262 pages, Publisher: Stationery Office; Version 3 edition, May 31, 2007, http://www.itil-itsm-world.com/servo.htm.

[Cass00] Cassidy, A., Guggenberger, K.: *A Practical Guide to Information Systems Process Improvement*, 288 p., ISBN 1-5744-4281-3, CRC Press, September, 2000.

[Cher06] Cherkasova, L. and Rolia, J.: *R-Opus: A Composite Framework for Application Performability and QoS in Shared Resource Pools*, in Proceedings of the 36th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'06), 2006.

[Cher84] Cheriton, D.R., *The V Kernel: A Software Base for Distributed Computing*, IEEE Software, pages 19-42, 1984.

[Cho82] Choo, Y.: *Hierarchical Nets: A Structured Petri Net Approach to Concurrency*, Technical Report CaltechCSTR:1982.5044-tr-82, California Institute of Technology, 1982.

[CIM] Distributed Management Task Force (DMTF), *Common Information Model (CIM)*, http://www.dmtf.org/standards/cim.

[CIMPol] DMTF, *CIM Policy*, http://www.dmtf.org/standards/documents/CIM/CIM_Schema26/CIM_Policy26.pdf.

[CIMv2.7] DMTF, *CIM Schema: Version 2.7*, http://www.dmtf.org/standards/cim/cim_schema_v27.

[CMIP] IETF: OSI, *Common Management Information Protocol (CMIP)*, RFC 1189, http://tools.ietf.org/html/rfc1189.

[COB] Information Systems Audit and Control Association (ISACA): *Control OBjectives for Information and related Technology (COBIT)*, http:// www.isaca.org/cobit/.

[Col05] Coleman, D., Cook, N., Eidt, E., Fleck, J., Graupner, S., Mukerji, J., Singhal, S., Thompson, C.: *Specification of the Service Delivery Controller (SDC)*, Hewlett-Packard, Software Global Business Unit, Cupertino, July 2005.

[Cond96] Epema, D.H., Livny, M. van Dantzig, R, Evers, X., Pruyne, J., *A Worldwide Flock of Condors: Load Sharing among Workstation Clusters*, Journal on Future Generations of Computer Systems, Volume 12, 1996 The Condor Project, http://www.cs.wisc.edu/condor/publications.html.

[Cook06] Cook, N., Graupner, S., Coleman, D., Fowler, C., Sarni, J.: *IT Utility Services Using Model-based Automation, Service-Oriented Architecture and Grid*, paper at HP TechCon 2006, Los Angeles, California, April 2-5, 2006.

[Cook06a] Cook, N., Coleman, D., Graupner, Sarni, J., Singhal, S., Thompson, C.: *Applying Service Delivery Controller in a Blade Automation Case*, poster at HP TechCon 2006, Los Angeles, California, April 2-5, 2006.

[Cor65] F. J. Corbató, V. A. Vyssotsky: *Introduction and Overview of the Multics System*, AFIPS Conf Proc 27, 185-196, 1965, http://www.multicians.org/fjcc1.html.

[CORBA] Object Management Group: *Common Object Request Broker Architecture (CORBA)*, http://www.corba.org.

[CPN] *CPN Tools*, http://wiki.daimi.au.dk/cpntools/cpntools.wiki.

[DCE] OpenGroup, *Distributed Computing Environment (DCE)*, http:// www.opengroup.org/dce.

[DCML] Oasis, *Data Center Markup Language (DCML)*, http://www.dcml.org.

[DCOM] Microsoft, *Distributed Component Object Model (DCOM)*, Remote Protocol Specification, http://msdn.microsoft.com/library/cc201989.aspx.

[Deit03] Deitel, H.M., Deitel, P.J., Choffnes, D.R.: *Operating Systems (3rd Edition)*, 1209 p., ISBN 0-1318-2827-4, December 2003.

[Den68] Peter J. Denning: *The Working Set Model for Program Behavior*, Communications of the ACM, Volume 11, Issue 5, pages: 323-333, May 1968.

[DFS] Silberschatz, Galvin. Operating System concepts, chapter 17, *Distributed File System* (DFS), Addison-Wesley Publishing Company, 1994.

[Dij68] Dijkstra, E.W.: *Cooperating Sequential Processes*, Programming Languages (F. Genuys, ed.), pages 43-112, Academic Press, London and New York, 1968.

[Dij71] Dijkstra, E.W.: *Hierarchical Ordering of Sequential Processes*, Acta Informatica, 1(1):115-138, January 1971.

[DMTF] *Distributed Management Task Force (DMTF)*, http://www.dmtf.org.

[Dre02] Dreo Rodosek, G.: *A Framework for IT Service Management*, Ludwig–Maximilians-Universität München, Habilitation, June, 2002.

[Duj99] J. J. Dujmovic, *Universal Benchmark Suites*, Proceedings of IEEE MASCOTS Conference, pp. 197-205, 1999.

[Edw07] Edwards, N., Belrose, G., Brand, K., Graupner, S., Rolia, J., Wilcock, L.: *Adaptive Infrastructure Meets Adaptive Applications*, Proceedings of the 14th HP OpenView University Association Conference (HP-OVUA), pp. 41-50, Munich, Germany, July 8-11, 2007.

[EGA] Enterprise Grid Alliance: *The EGA Reference Model*, 2005, http://www.gridalliance.org/en/WorkGroups/referencemodel_request.asp.

[eTOM] TeleManagement Forum: *Enhanced Telecom Operations Map (eTOM)*, http://www.tmforum.org/BusinessProcessFramework/1647/home.html.

[Fel99] Felfernig, A., Friedrich, G.E., et al., *UML as a Domain-specific Knowledge for the Construction of Knowledge-based Configuration Systems*, In the Proceedings of SEKE'99 Eleventh International Conference on Software Engineering and Knowledge Engineering, 1999.

[FML] Hewlett-Packard, *Farm Markup Language FML*, internal specification, 2002.

[Fost03] Foster, I., Kesselman, C.: *The Grid 2: Blueprint For A New Computing Infrastructure*, 2nd Edition, 748 p., ISBN: 1-55860-933-4, Morgan Kaufmann Publishers Inc, 2003.

[Fost98] Foster, I., Kesselman, C.: *The Grid: Blueprint For A New Computing Infrastructure*, 677 p., ISBN:1-55860-475-8, Morgan Kaufmann Publishers Inc, 1998.

[Gal00] Galis, A.: *Multi-Domain Communication Management*, ISBN 0-8493-0587-X, CRC Press LLC, Boca Raton, Florida, 2000.

[Gma07] Daniel Gmach, Jerry Rolia, Ludmila Cherkasova, Alfons Kemper: *Capacity Management and Demand Prediction for Next Generation Data Centers*. ICWS 2007: 43-50, 2007.

[Gma08] Gmach, D., Rolia, J., Cherkasova, L., Belrose, G., Turicchi, T., Kemper, A.: *An Integrated Approach to Resource Pool Management: Policies, Efficiency and Quality Metrics*, Proceedings of the 38th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'08), Anchorage, Alaska, June 24 - 27, 2008.

[Gma09] Gmach, D.: *Managing Shared Resource Pools for Enterprise Applications*, Ph.D. Dissertation, Technische Universität München, 2009.

[Gma10] Gmach, D., Chen, Y, Hyser, C., Wang, Z., Bash, C., Hoover, C., Singhal, S.: Integrated Management of Application Performance, Power and Cooling in Data Centers, 2010 Network Operations and Management Symposium (NOMS 2010), Osaka, Japan, April 19-23, 2010.

[Gol04] Goldsack, P.: *Smart Framework for Object Groups*, http://www.smartfrog.org, 2004.

[Gosc91] Goscinski, A.: *Distributed Operating Systems – The Logical Design*, 913 p., ISBN 0-201-41704-9, Addison-Wesley, July 1991.

[Gra97] Graupner, S.: *A Coherent Architecture of Operating Systems*, Ph.D. Dissertation, Chemnitz University of Technology, 198 p., published in Shaker Verlag, Aachen, ISBN 3-8265-3261-9, December 1997.

[Gra01] Graupner, S., Kotov, V., Trinks, H.: *Massive Deployment of Management Agents in Virtual Data Centers*, HP Labs Technical Report, HPL-2001-321, December, 2001.

[Gra02] Graupner, S., Kotov, V., Trinks, H.: *Resource-Sharing and Service Deployment in Virtual Data Centers*, The 22nd International Conference on Distributed Computing Systems Workshops (ICDCS 2002), pp. 666-671, Vienna, Austria, July 2-5, 2002.

[Gra03] Graupner, S., König, R., Machiraju, V., Pruyne, J., Sahai, V., van Moorsel, A.: *Impact of Virtualization on Management Systems*, 10th Workshop of the HP OpenView University Association (HP-OVUA), University of Geneva, Switzerland, July 6-9, 2003.

[Gra03a] Graupner, S.; Chevrot, J.-M.; Cook, N.; Kavanappillil, R.; Nitzsche, T.: *Adaptive Control System for Server Groups in Enterprise Data Centers*, HPL-2003-273, 2003.

[Gra03b] Graupner, S., Singhal, S.: *Conceptual Architecture of Quartermaster*, HP internal, architectural specification, April 2003.

[Gra04] Graupner, S., Nitzsche, T.: *Using HP's Web Services Management Framework for Adaptive Control*, HP TechCon 2004, Orlando, Florida, June 20-23, 2004.

[Gra04a] Graupner, S., Pruyne, J., Singhal, S.: *HP – Making the Utility Data Center A Power Station for the Enterprise Grid*, GRIDtoday, No. 740114, Vol. 3, No. 35, August 30, 2004.

[Gra04b] Graupner, S., Sahai, A.: *Web Services in the Enterprise: Concepts, Standards and Management*, Springer Verlag, ISBN 0-387-23374-1, 310 Seiten, 2004.

[Gra04c] Graupner, S., Chevrot, J.-M., Cook, N., Kavanappillil, R., Nitzsche, T.: *Adaptive Control for Server Groups in Enterprise Data Centers*, 4th IEEE/ACM Int. Symposium on Cluster Computing and the Grid (CCGrid 2004), Chicago, USA, April 19-22, 2004, also as HPL-2003-273.

[Gra04d] Graupner, S., Cook, N., Chevrot, J.-M., Kavanappillil, R.: *OGSI-based Adaptive Control System for the Utility Data Center*, Globusworld 2004, San Francisco, January 20 - 23, 2004.

[Gra04e] Graupner, S., Nitzsche, T.: *Web Services-based Management for Adaptive Control*, HPL-2004-94, May 20, 2004.

[Gra05] Graupner, S., Sahai, A.: *Policy-based Resource Topology Design for Enterprise Grids*, 5th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2005), Cardiff, UK, May 9-12, 2005; also appeared as HPL-2005-59 Technical Report, 2005.

[Gra05a] Graupner, S., Nitzsche, T.: *Extending Radia Into A Service Delivery Controller*, HP TechCon 2005, Phoenix, Arizona, March 20-23, 2005; also appeared as HPL-2005-52, March 11, 2005.

[Gra05b] Graupner, S., Nitzsche, T.: *Model-driven Software Configuration With the Radia SDC*, Proceedings of the 12th HP OpenView University Association Conference (HP-OVUA), pp. 397-400, Porto, Portugal, July 10-13, 2005.

[Gra05c] Graupner, S., Andrzejak, A., Kotov, V., Trinks, H.: *Adaptive Service Placement Algorithms for Autonomous Service Networks*, in Brueckner, S., Di Marzo Serugendo, G., Karageorgos, A., Nagpal, R. (Eds.): "Engineering Self-Organizing Systems", pages 280-298, LNCS 3464, Springer Verlag, May 2005.

[Gra06] Graupner, S., Cook, N., Coleman, D., Nitzsche, T.: *Management Middleware for Enterprise Grids*, 6th IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2006), Singapore, May, 2006.

[Gra06a] Graupner, S., Cook, N., Coleman, D., Nitzsche, T.: *Platform for Delivering IT Management Services*, COMSWARE 2006, New Delhi, India, January 8-12, 2006.

[Gra07] Graupner, S., Cook, N., Coleman, D.: *Automation Controller for Operational IT Management*, the 10th IFIP/IEEE Symposium on Integrated Management (IM 2007), Munich, Germany, May 21-25, 2007.

[Gra08] Graupner, S., Jerry Rolia, Nigel Edwards: *Deriving IT Configurations from Business Processes*. CEC/EEE 2008: 317-322, 2008.

[Gra08a] Graupner, S.: *ModelWeaver – A Service Design Environment for SAP*, Hewlett-Packard internal document, 2008.

[Grac96] R. Grace, *The Benchmark Book*, Prentice Hall, 1996.

[GT4] *The Globus Toolkit GT4*, http://www.globus.org.

[Hab76] Habermann, A.N., Flon, L., Cooprider, L.: *Modularization and Hierarchy in a Family of Operating Systems*, Communications of the ACM, 19(5):266-272, May 1976.

[Hal04] Halpern, J., Ellesoson, E.: *IETF Policy Framework*, concluded working group in IETF, 2004, http://www.ietf.org/wg/concluded/policy.html.

[Han01] Per Brinch Hansen: *Classic Operating Systems: From Batch Processing to Distributed Systems*, Springer; 1 edition, January, 2001.

[Härt90] Härtig, H., Kühnhauser, W., Kowalski, O., Lux, W., Reck, W., Streich, H., Goos, G.: *Architecture of the BirliX Operating System*, Technical Report, German National Research Center for Computer Science (GMD), Birlinghoven, March 1990.

[Heg94] Hegering, H.G., Abeck, S.: *Integrated Network and System Management*, Addison Wesley, 1994.

[Heg99] Hegering, H.G., Abeck, S., Neumair, B.: *Integrated Network of Networked Systems: Concepts, Architectures and their Operational Application*, The Morgan Kaufmann Series in Networking, 1999.

[Heg00] Hegering, H.G., Dreo Rodosek, G.: *Enterprise Management: Ganzheitliche Sicht auf das IT–Management in Unternehmen*, In Enterprise Networks and Call Centers, Congressband III, C320.1–C320.13, Online Verlag, Düsseldorf, February, 2000.

[Hell01] Hellerstein, J.L., Gandhi, N., Parekh, S.S.: *Managing the Performance of Lotus Notes: A Control Theoretic Approach*, International CMG Conference, 397-408, 2001.

[Hell04] Hellerstein, J.L., Diao, Y., Parekh, S.S., Tilbury, D.: *Feedback Control of Computing Systems*, John Wiley & Sons, New York, 2004.

[Hen89] van Hentenryck, P., *Constraint Satisfaction in Logic Programming*, The MIT Press, Cambridge, Mass, 1989.

[Herz01] Herzog, U., Rolia, J.: *Performance Validation Tools for Software / Hardware Systems*, Performance Evaluation 904 (2001), 1-22.

[Hin04] T. Hinrichs, N. Love, C. Pertie, L. Ramshaw, A. Sahai, S. Singhal: *Using Object Oriented Constraint Satisfaction for Automated Configuration Generation*. the 15th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM 2004), Davis, CA, USA, November 15-17, 2004.

[Hoa78] Hoare, C.A.R.: *Communicating Sequential Processes*, Communications of the ACM, 21(8):666-677, August 1978.

[Hol00] Holtsnider, B., Jaffe, B.D.: *IT Manager's Handbook: Getting Your New Job Done*, 337 pages, ISBN 1-5586-0646-7, Morgan Kaufmann; October, 2000.

[Hölz09] Hölzle, U., Barroso, L.A.: *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*, Google, Synthesis Digital Library of Engineering and Computer Science, Synthesis Lectures on Computer Architecture, Morgan&Claypool Publishers, 2009. http://www.morganclaypool.com.

[HP08] Hewlett-Packard: *Transforming Capacity Planning in Enterprise Data Centers: Enabling real-time Analysis and Optimization of Server Capacity and Power Use*; http://docs.hp.com/en/15052/CapPlan4AA1-9758ENW.pdf.

[HPAEM03] Hewlett-Packard, *HP Management Solutions for the Adaptive Enterprise*, Whitepaper, Hewlett-Packard, 2003, http://h71028.www7.hp.com/enterprise/cache/7504-0-0-0-121.aspx.

[HPBA03] Hewlett-Packard: *The HP Vision for the Adaptive Enterprise: Achieving Business Agility*, Whitepaper, 2003, http://h71028.www7.hp.com/enterprise/cache/7504-0-0-0-121.aspx.

[HPSAS10] HP Server Automation Software, https://h10078.www1.hp.com/cda/hpms/display/main/hpms_content.jsp?zn=bto&cp=1-11-271-273^14711_4000_100, 2010.

[Hri99] C. E. Hrischuk, M. Woodside, and J.A. Rolia, *Trace-based load characterization for generating performance software models*, Hrischuk, C.E.; Murray Woodside, C.; Rolia, J.A., IEEE Transactions on Software Engineering, vol. 25, no. 1, pp. 122-135, January-February 1999.

[Hub91] Huber, P., Jensen, K., Shapiro, R.M.: *Hierarchies in Colored Petri Nets*, In: G. Rozenberg (ed.): Advances in Petri Nets 1990, Lecture Notes in Computer Science Vol. 483, pages 313-341, Springer-Verlag, 1991.

[Hur05] Hurd, M.: *Automation Key to IT Cost Savings*, Computerworld, December 2005, http://www.computerworld.com/s/article/107029/Automation_key_to_IT_cost_savings_says_HP_s_Hurd.

[IDC03] IDC: *Enabling Business Agility: Hewlett-Packard's Adaptive Enterprise Strategy*, IDC Whitepaper, May 2003.

[IDS] IDS Scheer: *Aris*, http://www.ids-scheer.com.

[IETF] *Internet Engineering Task Force (IETF)*, http://www.ietf.org.

[IETFPol] IETF: *IETF Policy*, http://www.ietf.org/html.charters/policy-charter.html.

[Iqb07] Iqbal, M., Nieves, M.: *Service Strategy*, Vol 1, ITIL Version 3, 276 pages, Publisher: Stationery Office; Version 3 edition, Publisher: TSO, 2007, http://www.itil-itsm-world.com/servs.htm.

[ISO2K] *ISO/IEC 20000*, international standard for IT Service Management, http:// 20000.fwtk.org/.

[ISO9K] *ISO 9001:2008 Quality management systems – Requirements*. http://www.iso.org.

[ITIL] *IT Infrastructure Library (ITIL)*, http://www.itil.co.uk., http://www.itil-itsm-world.com.

[ITSM] *IT Service Management, The ITIL and ITSM Directory*, http://www.itil-itsm-world.com.

[itSMF] *The IT Service Management Forum*, http://www.itsmfi.org.

[J2EE] Sun Microsystems, *Java 2 Platform, Enterprise Edition (J2EE)*, http://java.sun.com/javaee.

[Jac09] Jackson, K.: *The Dawning of the IT Automation Era*, IT Business Edge, November 10, 2009, http://www.itbusinessedge.com/cm/community/features/guestopinions/blog/the-dawning-of-the-it-automation-era/?cs=37375.

[Jen97] K. Jensen: *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use*, Volumes 1-3, Monographs in Theoretical Computer Science, Springer-Verlag, ISBN: 3-540-60943-1, 2nd corrected printing, 1997.

[Jen98] Jensen, K.: *An Introduction to the Practical Use of Coloured Petri Nets*, In: W. Reisig and G. Rozenberg (eds.): Lectures on Petri Nets II: Applications, Lecture Notes in Computer Science vol. 1492, pages 237-292, Springer-Verlag 1998.

[Joch05] Jochum, C.: *Intelligent IT Sourcing in the Financial Industry: Background, Preconditions and Requirements of Future IT Organization Design*, Chapter in: The Practical Real-Time Enterprise, Springer Verlag Berlin Heidelberg, ISBN 9783540219958, http://www.springerlink.com/content/l83562nj48245026, December 2005.

[Kaa95] Kaashoek, M.F., Engler, D.R., O'Toole, J.: *Exokernel: An Operating System Architecture for Application-Level Resource Management*, Proceedings of the 15th SOSP, pages 251-266, December 1995.

[Kal90] Kalfa, W.: *Betriebssysteme*, 400 pages., Akademie Verlag, Berlin, 2. Auflage, 1990.

[Kell06] Keller, A., Diao, Y.: *Quantifying the Complexity of IT Service Management Processes*. DSOM 2006: 61-73.

[Kep03] Kephart, J., Chess, D.M.: *The Vision of Autonomic Computing*, IEEE Computer 36(1), 41-50, 2003. http://researchweb.watson.ibm.com/ autonomic.

[Kön04] König, R.: *Resource Management for a Federated Resource Utility*, Diploma (M.S.) Thesis, 95 pages, Chemnitz University of Technology with Hewlett-Packard Laboratories, Palo Alto, USA, January 2004.

[Kri00] U. Krishnaswamy and D. Scherson, *A Framework for Computer Performance Evaluation using Benchmark Sets*, IEEE Transactions on Computers, vol. 49, no. 12, pp. 1325-1338, December 2000.

[Kri04] D. Krishnamurthy, *Synthetic Workload Generation for Stress Testing Session-Based Systems*, PhD thesis, Dept. of Systems and Computer Eng., Carleton Univ., Ottawa, Canada, Jan. 2004.

[Kri06] D. Krishnamurthy, J. Rolia, and S. Majumdar, *A Synthetic Workload Generation Technique for Stress Testing of Session-based Systems*, IEEE Transactions on Software Engineering, vol. 32, no. 11, pp. 868-882, November 2006.

[Kri08] D. Krishnamurthy, J. Rolia, and M. Xu, *WAM - The Weighted Average Method for Predicting the Performance of Systems with Bursts of Customer Sessions*, HP Labs Technical Report, HPL-2008-66.

[Lac07] Lacy, S, Macfarlane, I.: *Service Transition*, Vol 3, ITIL Version 3, 261 pages, Publisher: Stationery Office; Version 3 edition, May 31, 2007, http://www.itil-itsm-world.com/servt.htm.

[Laz81] Lazowska, E.D., Levy, H.M., Almes, G.T., Fisher, M.J., Fowler, R.J., Vestal, S.C., *The Architecture of the Eden System*, Proceedings of the 8th SOSP, pp.148-159, Pacific Crove, California, 1981.

[LDAP] IETF, *Lightweight Directory Access Protocol (LDAP)*, RFC 1777, http://tools.ietf.org/html/rfc1777.

[Ley01] Leymann, F.: *Business Process Execution Language for Web Services (BPEL)*. Leymann, F.: Web Services Flow Language (WSFL), 2001.

[Lie95] Liedtke, J.: *On μ-Kernel Construction*, Proceedings of the 15th SOSP, pages 237-250, December 1995.

[Liu05] Liu, X., Zhu, X., Singhal, S., Arlitt, M.: *Adaptive Entitlement Control of Resource Containers on Shared Servers*, IFIP/IEEE International Symposium on Integrated Network Management (IM 2005), Nice, France, May 2005.

[LSF] Platform, Inc., *Load Sharing Facility LSF*, http://www.platform.com/products/LSF.

[Maui] *Maui Scheduler*, http://www.supercluster.org/maui.

[MIB] IETF, *Management Information Base (MIB)*, RFC 3418, http://tools.ietf.org/html/rfc3418.

[MOF] *Microsoft Operations Framework (MOF)*, http://www.microsoft.com/MOF.

[MPI] *Message Passing Interface*, http://www.mcs.anl.gov/mpi.

[MU-WS] OASIS: *Management Using Web Services (MUWS 1.0)*, Working Draft September 29 2004.

[N1] Sun Microsystems, *The Sun N1 Grid Engine*, http://wwws.sun.com/gridware.

[Nai04] Naik, V.K., Mohindra, A., Bantz, D.F.: *An Architecture for the Coordination of System Management Services*, p. 78-95, IBM Systems Journal, Vol. 43, No. 1, 2004.

[NGOSS] *NGOSS SLA Management Handbook*, TeleManagement Forum (TMF), 2005.

[NOW95] Anderson, T.E., Culler, D.E., Patterson, D.A.: *A Case for Networks of Workstations*: NOW. IEEE Micro, February 1995.

[OASIS] *The Organization for the Advancement of Structured Information Standards (OASIS)*, http://www.oasis-open.org.

[OCL] IBM, *Object Constraint Language (OCL)*, http://www.ibm.com/software/awdtools/library/standards/ocl.html.

[OGF] *The Open Grid Forum* (OGF), http://www.gridforum.org.

[OGSA] Foster, I., Kesselman, C., Nick, J., Tuecke, S., *The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration*, 2002, http://www.globus.org/research/papers/ogsa.pdf.

[OGSI] Global Grid Forum: *Open Grid Services Infrastructure (OGSI)*, v1.0, April 2003.

[OMG] *Object Management Group* (OMG), http://www.omg.org.

[OSA] The Parlay Group, *Open Services Architecture (OSA)*, ETSI OSA Parlay 3.0 Specifications, http://portal.etsi.org/docbox/TISPAN/Open/OSA/ParlayX30.html.

[OSF] *Open Software Foundation (OSF)*, now OpenGroup, http:// www.opengroup.org.

[Pan01] Pande, Peter S.; Neuman, Robert P.; Cavanagh, Roland R.: *The Six Sigma Way: How GE, Motorola, and Other Top Companies are Honing Their Performance*. New York: McGraw-Hill Professional, 229 pages, ISBN 0071358064, 2001.

[Parlay] *The Parlay Group*, http://www.parlay.org.

[PARPol] *PARLAY Policy Management*, http://www.parlay.org/specs.

[Pat03] Patel, C., Janakiraman, J., Bash, C., Farkas, K., Graupner, S.: *The Smart Data Center*, Proceedings of HP TechCon 2003, Keystone, Colorado, April 27-30, 2003.

[Pat03a] Patel, C., Sharma, R., Bash, C., Graupner, S.: *Energy Aware Grid: Global Workload Placement Based on Energy Efficiency, Int. Mechanical Engineering Congress and Exhibition (IMECE-2003)*, Washington DC, Nov 16-21, 2003.

[PBS] *Portable Batch System PBS*, http://www.openpbs.com.

[Pegasus] The Open Group: *OpenPegasus*, http://www.openpegasus.org.

[Petr07] D. C. Petriu, C. M. Woodside, D. B. Petriu, J. Xu, T. Israr, G. Georg, R. France, J. M. Bieman, S. H. Houmb, and J. Jurjens, *Performance Analysis of Security Aspects in UML Models*, Proceedings of the 6th ACM International Workshop on Software and Performance (WOSP07), pp 91-102, Buenos Aires, February 2007.

[Petri62] Petri, C.A.: *Kommunikation mit Automaten*, Ph.D. Dissertation, University of Bonn, Germany 1962.

[Pett81] Petterson, J.L.: *Petri Net Theory and the Modeling of Systems*, Prentice-Hall, Englewood Cliffs, 1981.

[Pop85] Popek, G., Walker, B.J.: *The LOCUS Distributed System Architecture*, The MIT Press, Cambridge, Mass., 1985.

[PRM-IT] *The IBM Process Reference Model for IT (PRM-IT)*, http://www-01.ibm.com/software/tivoli/governance/servicemanagement/welcome/process_reference.html.

[PVM] *Parallel Virtual Machine (PVM)*, http://www.epm.ornl.gov/pvm.

[RAD] Novadigm, Inc.: *Radia Configuration Management*, 2004.

[Ram06] Ramshaw, L., Sahai, A., Saxe, J., Singhal, S.: *Cauldron: A Policy-based Design Tool*, 2006 IEEE Workshop on Policies for Distributed Systems and Networks (Policy 2006). The University of Western Ontario, Canada, June 5-7, 2006.

[Ram98] Raman, R., Livny, M., Solomon, M., *MatchMaking: Distributed Resource Management for High Throughput Computing*, In the proceedings of HPDC 98.

[Rash86] Rashid, R.F.: *Experiences with the Accent Network Operating System*, in Miller, G., Blanc, R.P. Networking in Open Systems, LNCS 248, pages 259-269, Springer, 1986.

[Rei79] M. Reiser, *A Queuing Network Analysis of Computer Communication Networks with Window Flow Control*, IEEE Transactions on Communications, vol. 27, no. 8, pp. 1201-1209, August 1979.

[RFC 3703] IETF, *Policy Core Lightweight Directory Access Protocol (LDAP) Schema*, RFC 3703, http://tools.ietf.org/html/rfc3703.

[RFC2608] IETF, *Service Location Protocol, Version 2*, RFC 2608, http://tools.ietf.org/html/rfc2608.

[RFC2753] IETF, *A Framework for Policy-based Admission Control*, RFC 2753, http://tools.ietf.org/html/rfc2753.

[RFC3060] IETF, *Policy Core Information Model*, RFC 3060, http://tools.ietf.org/html/rfc3060.

[Rob04] Robison, S., *Grids for the Enterprise*, Grid.Middleware Spectra, 3rd Edition, Spring 2004.

[Rol95] Rolia, J., Sevcik, K.C.: *The Method of Layers*, IEEE Transactions on Software Engineering, vol. 21, no. 8, pp. 689-700, August 1995.

[Rol98] Rolia, J., Vetland, V.: *Correlating Resource Demand Information with ARM Data for Application Services*, Proceedings of the ACM Workshop on Software and Performance (WOSP '98), pages 219-230, Santa Fe, NM, USA, October 1998.

[Rol04] Rolia. J., Cherkasova, L., Arlitt, M., Andrzejak, A.: *A Capacity Management Service for Resource Pools*. WOSP 2005: 229-237, 2005.

[Rol05] Rolia, J., et. al., *Capacity Management for Adaptive Enterprise Resource Pools*, HP TechCon 2005, Phoenix, Arizona, March 20-23, 2005.

[Rol06] Rolia, J., Cherkasova, L., Arlitt, M., Machiraju, V.: *Supporting Application QoS in Shared Resource Pools*, Communications of ACM, special issue on Self-managed Systems and Services Vol. 49, Issue 3, pages 55-60, March 2006.

[Rol08] Rolia, J., Krishnamurthy, D., Xu, M., Graupner, S.: *APE: An Automated Performance Engineering Process for Software as a Service Environments*, submitted to the Special Issue of IEEE Transactions on

Software Engineering on Quantitative Evaluation of Computer Systems, HPL-2008-65, 37 pages, June 7, 2008.

[RSL] Globus, *The Globus Resource Specification Language RSL v1.0*, http://www.globus.org/gram/rsl_spec1.html.

[Rud07] Rudd, C., Lloyd, V.: *Service Design*, Vol 2, ITIL Version 3, 334 pages, Publisher: Stationery Office; Version 3 edition, May 31, 2007, http://www.itil-itsm-world.com/servd.htm.

[Sad06] Sadiq, W, Sadqi, S., Schulz, K.: *Model Driven Distribution of Collaborative Processes*, IEEE International Conference on Services Computing (SCC 2006), Chicago, USA. Sep 2006.

[Sah03] Sahai, A., Graupner, S., Machiraju, V., van Moorsel, A.: *Specifying and Monitoring Guarantees in Commercial Grids through SLA*, Thef 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2003), pp. 292-299, Tokyo, Japan, May 12-15, 2003.

[Sah04] Sahai, S. Singhal, V. Machiraju, R. Joshi: *Automated Generation of Resource Configurations through Policies*, in IEEE 5th International Workshop on Policies for Distributed Systems and Networks (Policy 2004), Yorktown Heights, NY, June 7-9, 2004.

[Sah04a] Akhil Sahai, Sharad Singhal, Rajeev Joshi, Vijay Machiraju: *Automated Policy-Based Resource Construction in Utility Environments*, the 9th IEEE/IFIP Network Operations and Management Symposium (NOMS 2004). COEX Convention Center, Seoul, Korea, 19-23 April, 2004.

[Sah05] Sahai, A., Goswami, K., Pruyne, J., Singhal, S., Potts, M., Sparkes, A., Polakowski, K., Machiraju, V., Graupner, S.: *Virtual Desktop Initiative: Desktops as Services in a Utility Computing Environment*, HPL-2005-219, Dec 2005.

[Sah06] Sahai, A., Goswami, K., Blaho, B., Smith, M., Graupner, S., Hochmuth, R., Young, D., Sarni, D.: *Virtual Desktop System: Consolidating Enterprise User Desktops*, paper at HP TechCon 2006, Los Angeles, California, April 2-5, 2006.

[San04] Santos, A. Sahai, X. Zhu, V. Machiraju, S. Singhal: *Policy-based Resource Assignment in Utility Computing Environments*, the 15th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM 2004), Davis, CA, USA, November 15-17, 2004.

[San05] Santos, C., et. al.: *RAMP-A Solver for Automated Resource Assignment in Computing Utilities*, HP TechCon 2005.

[SAP08] *SAP Business by Design*, http://www.sap.com/solutions/sme/businessbydesign/index.epx.

[Sauv06] Sauve, J. P., Marques, F. T., Moura, J.A.B., Sampaio, M.C., Jornada, J., Radziuk, E.: *Business-Oriented Capacity Planning of IT Infrastructure to Handle Load Surges*. In: IEEE/IFIP Network Operations & Management Symposium - NOMS 2006, 2006, Vancouver. Proceedings of 10th IEEE/IFIP Network Operations & Management Symposium. Vancouver : IEEE, v. 10. p. 1-4., 2006.

[Sauv07] Sauve, J. P., Santos, R. A., Almeida, R. R., Moura, J.A.B.: *On the Risk Exposure and Priority Determination of Changes in IT Service Management*, In: The 18th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, DSOM 2007, 2007, San José, United States. Large Scale Management of Distributed Systems, Springer Verlag Berlin Heidelberg, 2007.

[Sauv09] Sauve, J. P., Moura, J.A.B., Marques, F. T.: *Business-Driven Design of Infrastructures for IT Services*, Journal of Network and Systems Management, Vol. 17, No. 4, http://www.springerlink.com/content/h271580kq8413443, December 2009.

[Scha07] Schaaf, T.: *Frameworks for Business-driven Service Level Management*, Second IEEE/IFIP International Workshop on Business-driven IT Management (BDIM 2007) in conjunction with IM 2007, Munich, Germany, May 21, 2007.

[Sche04] Scheer, A.W., Abolhassan, F., Jost, W., Kirchmer, M. (Eds.): *Business Process Automation*, ISBN 3540207945, Springer Verlag, 2004.

[Sche06] Scheer, A.W.: *Business Process Design*, Springer; 3 Edition, September 22, 2006.

[Schie01] Schiesser, R.: *IT Systems Management: Designing, Implementing, and Managing World-Class Infrastructures*, 528 p., ISBN 0-1308-7678-X, Prentice Hall, December 2001.

[Schie02] Schiesser, Rich: *IT Systems Management*, Prentice Hall, 2002.

[Schie04] Schiesser, Rich: *IT Management Reference Guide: Capacity Planning – Part One: Why It is Seldom Done Well*, http://www.informit.com/guides/content.aspx?g=it_management&seqNum=34.

[SF04] *The SmartFrog Reference Manual – A Guide to Programming with the SmartFrog Framework*, July 2004, http://www.smartfrog.org.

[Silb02] Silberschatz, A., Galvin, P.B., Gagne, G., *Operating System Concepts*, 6th Edition, 976 p., ISBN 0-4712-5060-0, John Wiley & Sons, March 2002.

[Sing04] Singhal, S., Graupner, S., Sahai, A., Machiraju, V., Pruyne, J., Zhu, X., Rolia, J., Arlitt, M., Santos, C., Beyer, D., Ward, J.: *Quartermaster - A Resource Utility*, HP TechCon 2004, Orlando, Florida, June 20-23, 2004, also appeared as HPL-2004-152, Sep 9, 2004.

[Sing05] Singhal, S., Graupner, S., Sahai, A., Machiraju, V.: *Quartermaster – A Resource Utility System*, IM 2005, Nice, France, May 15-19, 2005.

[Slo01] Sloman, M., Damianou, N., Dulay, N., Lupu, E., *The Ponder Policy Specification Language*, pages 18-38, POLICY 2001.

[Slo93] Sloman, M.J., *Policy Conflict Analysis in Distributed Systems*, In the proceedings of Journal of Organizational Computing, 1993.

[Slo94] Sloman, M.: *Network and Distributed Systems Management*, 666 p., Addison Wesley, June 1994.

[Smith04] Smith, B.F., Kreitz, J., Wilson, M.M.: *Autonomic IMS and IMS Tools*, The Mainstream, The IBM eServer zSeries and S/390 Software Newsletter, Issue 8, April 12, 2004.

[Smith90] C.U. Smith, *Performance Engineering of Software Systems*, Addison-Wesley, 1990.

[SML] *SML: The Service Modeling Language Specification*, v0.5, Draft Specification, July 2006. http://go.microsoft.com/fwlink/?LinkId=70293.

[Sne02] Snevely, R.: *Enterprise Data Center Design and Methodology*, Sun Blueprint Series, 224 pages, ISBN 0-1304-7393-6, Prentice Hall, January 2002.

[SNIA] *Storage Networking Industry Association (SNIA)*, http://www.snia.org.

[SNIA-OM] The Open Group: *SNIA CIM Object Manager*, http://www.opengroup.org/snia-cimom.

[SNMP] IETF, *Simple Network Management Protocol (SNMP)*, RFC 1157, http://tools.ietf.org/html/rfc1157.

[Spa07] Spalding, G.: *Continual Service Improvement*, Vol 5, ITIL Version 3, 221 pages, Publisher: Stationery Office; Version 3 edition, May 31, 2007, http://www.itil-itsm-world.com/cserv.htm.

[Stal04] Stallings, W.: *Operating Systems (5th Edition)*, 832 p., ISBN 0-1314-7954-7, Prentice Hall, July 2004.

[Stew07] C. Stewart, T. Kelly, and A. Zhang, *Exploiting non-Stationarity for Performance Prediction*, Proceedings of the EuroSys Conference, pp. 31-34, Lisbon, Portugal, March 2007.

[Stra03] Strassner, J.: *Policy-Based Network Management: Solutions for the Next Generation*, The Morgan Kaufmann Series in Networking, 2003.

[Sun99] X. Sun, *Estimating Resource Demands for Application Services*, M. Sc. Thesis, Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada, 1999.

[Tan91] Tanenbaum, A.S., Kaashoek, M.F., Renesse, R. van, Bal, H.: *The Amoeba Distributed Operating System*, Computer Communications, Vol. 14, pp. 324-335, July/August 1991.

[Tan94] Tanenbaum, A.S.: *Distributed Operating Systems*, 648 pages, Prentice Hall, 1994.

[Tan01] Tanenbaum, A.S.: *Modern Operating Systems (2nd Edition)*, 976 p., ISBN 0-1303-1358-0, Prentice Hall, February 2001.

[Thom05] Thompson, C., Coleman, D.: *Model Based Automation and Management for the Adaptive Enterprise*, 12th Annual Workshop of HP OpenView University Association, Porto, Portugal, July 10-13, 2005.

[Tiw06] N. Tiwari and P. Mynampati, *Experiences of using LQN and QPN tools for performance modeling of a J2EE application*, International Computer Measurement Group (CMG) Conference, pp. 537-548, 2006.

[TMF] *TeleManagement Forum (TMF)*, http://www.tmforum.org.

[TOM] TM Forum (TMF), *Telecom Operations Map (TOM)*, http://www.tmforum.org.

[TQ04] TeamQuest Corp.: *Capacity Planning Discipline for Data Center Decisions*, http://www.teamquest.com/pdfs/whitepaper/tqeb01.pdf, 2004.

[UDC] Hewlett-Packard, *The Utility Data Center*, Hewlett-Packard, http://www.hp.com/go/udc.

[Ver01] Verma, D., Beigi, M., Jennings, R., *Policy-based SLA Management in Enterprise Networks*, Workshop on Policy, POLICY 2001.

[Vetl93] V. Vetland, *Measurement-based composite computational work modeling of software*, Ph.D. Thesis, University of Trondheim, August 1993.

[Vos99] Vose, M.D.: *The Simple Genetic Algorithm: Foundations and Theory*, MIT Press, Cambridge, MA, 1999.

[Wang05] Wang, Z., Zhu, X., Singhal, S.: *Utilization and SLO-Based Control for Dynamic Sizing of Resource Partitions*, Distributed Systems: Operations and Management Workshop (DSOM 2005), Barcelona, Spain, October 24-26, 2005.

[WBEM] DMTF, *Web-Based Enterprise Management (WBEM)*, http://www.dmtf.org/standards/wbem.

[Wels03] Welsh, M., Culler, D.: *Adaptive Overload Control for Busy Internet Servers*, 4th USENIX Symposium on Internet Technologies and Systems (USITS 2003), Seattle, March 2003.

[Wes07] Weske, M.: *Business Process Management: Concepts, Languages, Architectures*, 368 pages, ISBN 978-3-540-73521-2, Springer Verlag, 2007.

[Wett93] Wettstein, H.: *Systemarchitektur*, Hanser Studienbücherei der Informatik, Carl Hanser Verlag, MünchenWien, 1993.

[Wood95] M. Woodside, J. E. Nielsen, D. C. Petriu, and S. Majumdar, *The stochastic rendezvous network model for performance of synchronous client-server-like distributed software*, IEEE Transactions on Computers, vol. 44, no. 1, pp, 20-34, January 1995.

[WS-DM] OASIS: *Web Services Distributed Management (WSDM),* http://www.oasis-open.org/specs, Working Draft September 29 2004.

[WS-MAN] *Web Services for Management (WS-Management)*, v1.0, April 2006, http://www.dmtf.org/standards/wsman.

[WS-Pol] OASIS, *WS-Policy WG*, http://www.oasis-open.org.

[WS-RF] OASIS TC and Global Grid Forum, *The Web Services Resource Framework (WSRF)*, April 2004, http://www.globus.org/wsrf.

[xmlCIM] Distributed Management Task Force: *WBEM: CIM-XML*, http://www.dmtf.org/standards/wbem/CIM-XML.

[Xu06] Xu, W., Zhu, X., Singhal, S., Wang, Z.: *Predictive Control for Dynamic Resource Allocation in Enterprise Data Centers*, 2006 IEEE/IFIP Network Operations and Management Symposium (NOMS 2006), Vancouver, Canada, April 3-7, 2006.

[Zhang07] Q. Zhang, L. Cherkasova, and E. Smirni, *A regression-based analytic model for dynamic resource provisioning of multi-tier applications*, 4th International Conference on Autonomic Computing, pp. 27-27, June 2007.

[Zhu01] Zhu, X., Singhal, S.: *Optimal Resource Assignment in Internet Data Centers*, 9th International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS 2001), Cincinnati, August 15-18, 2001.

[Zhu03] Zhu, X., Santos, C., Ward, J., Beyer, D., Singhal, S.: *Resource Assignment for Large Scale Computing Utilities*, HP Labs Technical Report, HPL-2003-243, 2003.

[Zhu04] Zhu, X., Santos, C., Beyer, D., Singhal, S., *Extension of the Resource Assignment Problem: Assigning Multiple Application Components to a Single Server in a Generalized LAN Tree Topology*, HP Labs Technical Report, HPL-2004-42, 2004, http://www.hpl.hp.com/techreports/2004/HPL-2004-142.html.

[Zhu08] Zhu. X., Young, D., Watson, B., Wang, Z., Rolia, J., Singhal, S., McKee, B., Hyser, C., Gmach, D., Gardner, R., Christian, T., Cherkasova, L.: *1000 Islands: Integrated Capacity and Workload Management for the Next Generation Data Center*, HP Labs (internal) Technical Report, HPL-2008-3, 2008, Proceedings of the 5th IEEE International Conference on Autonomic Computing (ICAC'08), July 2 - 6, 2008, Chicago, IL, USA.

References

List of Figures

List of Tables