CMDB Patterns

Designing CMDB data models with good utility and limited complexity

Michael Brenner; Markus Gillmeister
Leibniz Supercomputing Centre of the Bavarian Academy of Sciences and Humanities
Garching n. Munich, Germany

This paper introduces CMDB patterns as an approach to help address conceptual issues in CMDB implementations and provide practitioners with a common set of terms for useful designs.

Configuration Management Database (CMDB) is one of the most central concepts in IT Service Management (ITSM). the CMDB is a tool, maintained by the ITSM process *Configuration Management*, that provides information about *Configuration Items* (CI) which contribute to the delivery of an IT service, as well as the relationships between CIs and between CIs and IT services. Descriptions and discussions of the majority of ITSM processes defined in ITIL [1] or ISO/IEC 20000 [2] refer to the CMDB as source of information, vital for the process to function effectively.

978-1-4799-0913-1/14/$31.00 ©2014 IEEE

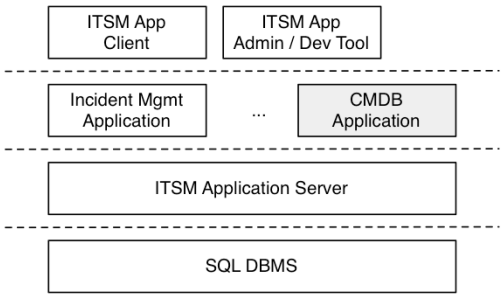
CMDB as a concept and as a tool

CMDB as a concept of ISO/IEC 20000 and ITIL

- *Configuration management database (CMDB)*
  Data store used to record attributes of configuration items, and the relationships between configuration items, throughout their lifecycle
- *Configuration item (CI)*
  Element that needs to be controlled in order to deliver a service or services

CMDB as a tool

- Usually part of a comprehensive ITSM suite
- Allows linking CIs to incident records, problem records, change records etc. and vice versa
- SQL basis usually quite noticeable (no true object orientation)

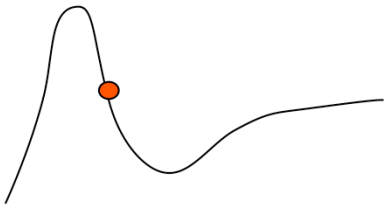


The leading ITSM publications and standards describe it in rather general terms. ISO/IEC 20000 defines a CMDB as *a data store used to record attributes of configuration items, and the relationships between configuration items, throughout their lifecycle* [1]
(Unfortunately, since its third versions published in 2007, the ITIL books have started use the term CMDB to denote a single database, while newly introduced concept *Configuration Management System* or *CMS* – a kind of „super CMDB“ which *includes tools for collecting, storing, managing, updating, analysing and presenting data about all configuration items and their relationships* – now serves the same purpose as the original CMDB concept [2]. For simplicity, we will stick to the term *CMDB* in its original meaning for the remainder of this paper.)

In practice, a CMDB is usually not a single database, but a tool that synchronizes and reconciles configuration information from various sources (*management data repositories*), and enables the mapping and visualization of CI-relationships [4]. As a piece of software, it needs to be integrated, and in most cases also shares a common platform, with other ITSM applications to form an *ITSM Suite*, that allows CIs to be linked to artifacts of other ITSM processes like incident records, problem records, change records etc.

Status quo of the CMDB in practice

- About half of all CMDB projects fail [Gartner 2013], “Cause of death” is almost always complexity
- Part of the IT service management community questions whether implementing and maintaining a CMDB is justifiable from a business point of view:
  It is such an enormous undertaking that any organisation attempting it is going to burn money on an irresponsible scale.
  (Blog post by IT Skeptic: “ITIL’s dead elephant: CMDB can't be done”)
- CMDB is currently heading down into the “trough of disillusionment” in the Gartner hype cycle.



Position of the “Service View CMDB” in the hype cycle for IT operations management

Despite its importance, the guidance of ISO/IEC 20000 and ITIL on implementing CMDBs (or CMSs) remains surprisingly vague.
As a consequence, the CMDB solutions that ITSM software vendors and ITSM practitioners come up with, differ quite significantly in scope, structure and content.

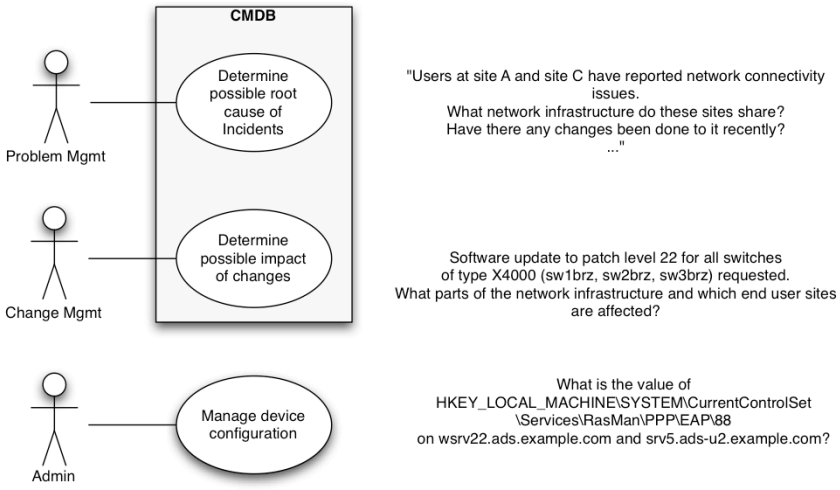
The expectations for what a CMDB should offer are often unrealistically high, resulting in too-ambitious projects of which quite a large portion fail [3], and leading some ITSM experts to question the practicality of the whole concept [6].
Gartner sees the CMDB currently heading downwards in its IT operations management hype cycle [5].

Clearly, more concrete guidance on implementing CMDBs is needed.

Towards a higher success rate for CMDB projects

- Status quo: Each organization designs its own CMDB information model. This is not likely to change in the near future!
  - Universally accepted common information and data model for CMDBs is nowhere in sight
  - Service providers' infrastructures, business models, company cultures etc. vary greatly, there might never be one CMDB information model to suit all needs
- What is needed:
  Non-prescriptive, adaptable, pragmatic guidance on CMDB design
  - Setting the right scope for a CMDB project
    -> clarification of CMDB requirements and prioritization of use cases
  - Guidance on designing the information model
    -> CMDB patterns

Setting the scope: CMDB use cases



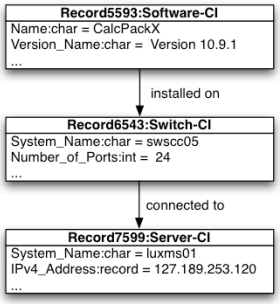
Goal: Benefit from finding the answers to these questions easier and quicker through use of the CMDB >> Cost of maintaining the CMDB

A very common obstacle towards a successful CMDB implementation are unrealistic expectations.
Most IT staff wish for access to better and more detailed documentation. As the CMDB concept is so vaguely defined, there is – almost as with a Rorschach inkplot – much room for interpretation and many envision a tool that will finally address all their documentation needs (and is thankfully provided, paid for and maintained by an ITSM project team).It is therefore important to manage these expectations and to clarify which requirements a CMDB solution will have to fulfill, and which functionalities are maybe nice-to-have, but non-essential.

The CMDB is a tool to be used in the context of ITSM processes. The use cases (or query cases) it needs to address, and which should be prioritized, are in the context of ITSM processes,, e.g. problem management or change management. Fulfilling all IT administrators' requirements for a documentation tool with a single solution of is an unachievable goal. It is important to convey that in most cases, introducing a CMDB solution at an IT service provider organization will replace only very few, if any, existing tools for documenting configuration information.

CMDB Patterns

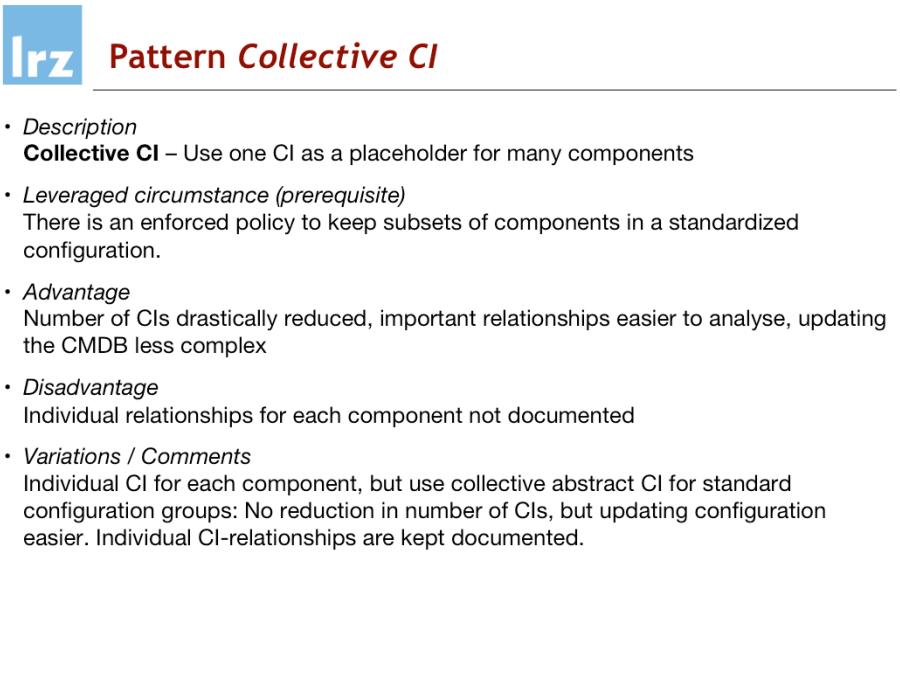
- Majority of CMDBs are composed of
  - CI records, derived from a template and containing a number of attributes of a simple type
  - Binary and directed CI-relationships
- CMDBs become more complex, the more
  - attributes the records contain;
  - CI records there are;
  - CI-relationships there are per CI.
- CMDB patterns should
  - help to design CMDB information models that offer a good utility/maintenance ratio;
  - allow CMDB designers to share and discuss ideas using a common terminology;
  - limit the complexity of the resulting CMDB.



There is an abundance of very varied software that claims to support Configuration Management or CMDB implementation.
Still, almost all leading commercial solutions share the same basic characteristics:
They support the definition of templates (or class definitions) for CI records, which can contain typed (int, char, boolean…) attributes. CI-relationships are almost always limited to be binary and directed, but can otherwise be freely defined or adapted. Samples are provided, but generally the definition of the CI record templates and CI-relationship types is up to the organization that wants to implement the CMDB.
This is a quite demanding task for most organizations, that do not specialize in ITSM, and usually requires extensive third party consulting.

Our goal is to introduce CMDB patterns – doing for CMDB design what Fowler's *Analysis Patterns* [7] did for the design of business information systems: Start to provide higher-order designs that can be reused across projects and types of infrastructures and thereby facilitate the future reuse, discussion and sharing of good CMDB design ideas.

In the following, we will discuss our first three patterns, which evolved while facing design issues during the development of a CMDB for services of the Leibniz Supercomputing Centre.



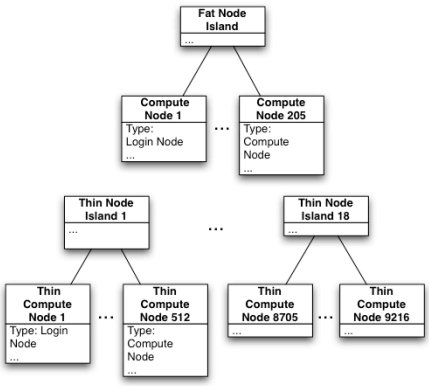
The first pattern is called *Collective CI* and is, of the patterns presented in this paper, probably the most commonly used.

The idea is simple: If sets of components are either kept at an identical configuration or are not configurable (e.g. keyboards, monitors…), a single CI (Collective CI) can act as a placeholder for many components.

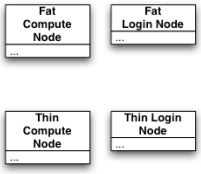
Of course, some information is lost when this pattern is used. If a set of components is configured identically, but the documentation of the relationships for each individual component is still essential, this pattern should not be used.
However, quite often the most important CMDB use cases can still be addressed using one CI for many components, and the reduction in complexity actually enables a simpler and more effective analysis.

Collective CI - Example: Supercomputer nodes

Common approach:
1 CI per addressable node
+ 1 CI per island
>9000 CIs



Use of *Collective CI* Pattern
4 CIs

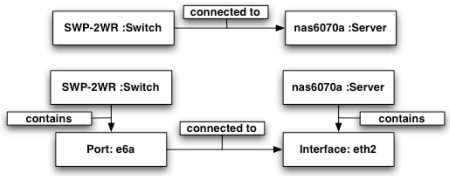


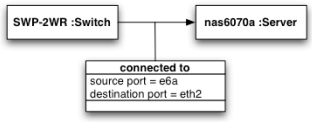
In this only slightly simplified example based on a real-world scenario (cp. http://www.lrz.de/services/compute/supermuc/systemdescription/), the *Collective CI* pattern is used to provide a very simple CMDB model of a supercomputer.
The first idea for creating a CMDB model of this supercomputer was to mimic the system architecture, creating one CI for each of the 900 hardware nodes and linking them to the CIs of the “islands” in which they are arranged.

However, all the nodes fall into one of just two hardware types, so-called *thin nodes* (with two 8-core processors each) and *fat nodes* (with four 10-core processors each). All nodes of each type boot from one of two software configurations, as a *Compute Node* or as a *Login Node*.
Consequently, for managing software-related incidents, problems, changes and releases, the nodes of each type are interchangeable. As hardware failures of individual nodes are relatively easy to diagnose and the nodes easily exchanges, only little value is gained from distinguishing identically configured nodes.
The model using the Collective CI pattern contains only 4 CIs compared to the over 9000 that would have been required for the more straight-forward approach. Still, the utility for the ITSM processes is almost as high and quite a few typical use cases – e.g. analysing if a number of similar incidents has occurred on all types of nodes or just one – are actually more easily addressed.

Pattern *Rich CI relations*

- *Description*
  Rich CI relations – Add editable attributes to CI-relations; preferably including *multi-value attributes*
- *Leveraged circumstance (prerequisite)*
  Data model allows attributes to be added to relationships (usually given for CMDB based on relational databases)
- *Advantage*
  Number of CIs drastically reduced, major relationships clearer
- *Possible Disadvantage*
  Individual network components (NICs etc.) no longer controlled as individual CIs
- *Variation / Comments*
  Pattern also useable for other documentation cases (e.g. mounting volumes)



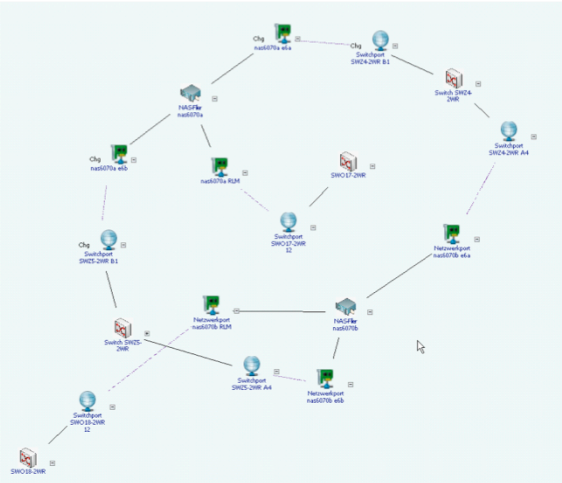


In most commercial CMDB solutions, CI-relationships are point-to-point (1-to-1) and are defined by direction (source, destination) and type („depends on“, „is part of“, „is backed up by“ etc.) only.

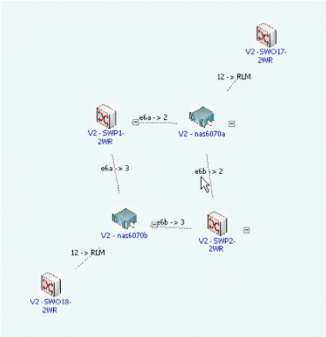
Especially for the documentation of complex network topologies this often results in models that are either very complex – or miss representing essential information.

Rich CI relations - Example

Model 1



Model 2



Above Screenshots show the visualization of two models of the same infrastructure, as rendered with an "auto layout" function of a commercial CMDB solution (iET Solutions CMDB).

A requirement was, that the physical interconnection – “which port of the switch is the connected to interface e6a of the NAS filer?” – should be documented in the model.
Model 1, created with the out-of-the-box data model of the CMDB solution, achieves this by defining NAS interfaces and switch ports as CIs and using simple (attribute-less) “network connection” relationships.
Model 2 uses an adapted data model, realizing the *rich CI relations* pattern, adding a “source port” and “destination port” attribute to the network connection relationship. It would also be relatively easy to add other information like VLAN numbers, link capacity etc. without introducing more CIs.

When visualized, model 2 is obviously simpler and more intuitive to understand.
One of the most interesting features of the modeled network topology – that the NAS-filers are connected redundantly via switches SWP1-2WR and SWP2-2WR – is more readily apparent by looking at the representation of model 2.

Pattern *Multi-value attributes*

- *Description*
  Multi-value attributes – Enable record-type attributes in CI records
- *Leveraged circumstance (prerequisite)*
  Data model supports an attribute record type or is adaptable to suit one (e.g. support for comma-separated lists in attributes)
- *Advantage*
  Modeling of associations between system sub-components – e.g. how are IP addresses, MAC addresses and network interfaces bound to each other – much more efficient than with multiple-CI solutions.
- *Possible Disadvantage*
  Depending on the data model and implementation of the new type, some queries become more complex (e.g.”Which IP addresses are currently unused?”)
- *Variation / Comments*
  Multi-value attributes are ideal for modeling the network configuration of systems, but are also useful for describing mass storage components.
  Multi-value attributes and *Rich CI relations* are distinct patterns, but serve the same purpose – reducing the number of CIs with no or little loss of information – and are best used in combination.

In out-of-the-box state, most CMDB solutions support CI records with simple attribute types (integer, char etc.) only. Modeling of interdependencies – e.g. “what IP addresses are bound to which MAC addresses?” – would require the creation of many more CIs (e.g. one CI for each network address on each layer) with many relationships.

Having multi-value attributes (aka an attribute type “record”) offers a much more efficient solution.

Typical applciations are the documentation of the network configuration (e.g. <DEV>;<MAC>;<IPv4>;<IPv6>;<DNS>) or the mass storage configuration (e.g. <TYPE>;<DEVICE>;<SIZE>;<MOUNTPOINT>) of server systems.

Multi-value attributes – Example

Approach 1:

The way most out-of-the box example models look like. Common queries remain simple, but associations are lost.

Server	
Sys-Name	luxms01
MAC	00:07:E9:23:CD:59
MAC	00:07:E9:23:CD:58
IPv4	127.189.253.120/24
IPv4	127.189.253.108/24
IPv4	127.189.253.111/24
IPv4	127.189.253.102/24
IPv4	127.189.253.100/24
IPv4	11.77.6.1/24
IPv4	11.77.6.51/24

Approach 1 (CIM-like):

Conceptually clean, but many CIs and relationships. Many common queries ("what server has IP addresses 127.189.253.120?") and consistency checks become difficult.

Approach 3 (multi-value attributes):

Associations are kept, common queries remain simple, while some others ("which IP-addresses are free?") become more complex and need to be predefined.

Server			
luxms01			
eth0	00:07:E9:23:CD:59	127.189.253.120/24	luxms01.example.com
eth0	00:07:E9:23:CD:59	127.189.253.108/24	relay4.example.com
eth0	00:07:E9:23:CD:59	127.189.253.111/24	-
eth0	00:07:E9:23:CD:59	127.189.253.102/24	relay2.example.com
eth0	00:07:E9:23:CD:59	127.189.253.100/24	relay6.example.com
eth1	00:07:E9:23:CD:58	11.77.6.1/24	-
eth1	00:07:E9:23:CD:58	11.77.6.51/24	relay2.mail.example.com

Limitations and future work

Limitations

- So far, very limited number of patterns.
- So far, practical demonstration of utility in only one scenario.
- *Rich CI relations* and *multi-value attributes* address specific weaknesses of common SQL-based CMDB data models. May become obsolete with better out-of-the-box CMDB data models.

Future work

Patterns are “best practice”: Spread the idea and encourage participation

- Promote adoption of *rich CI relations* and *multi-value attributes with* CMDB tool developers.
- Promote use of patterns by other CMDB practitioners (e.g. by integrating them in a FitSM Guide).
- Gather feedback and new ideas.
- Develop further patterns and refine existing ones.

CMDB patterns are documenting “good practice” (or “best practice”) in CMDB design.
In the long term, they should not remain the product of a small number of authors, but be used, discussed, refined and extended by a community of CMDB practitioners.

The first step in the further development of a CMDB pattern catalogue would therefore be promoting the use of existing patterns, and disseminating the “pattern idea” for CMDBs in general, e.g. by integrating CMDB patterns in a future guide on Configuration Management in FitSM-5 [8].

Conclusion

- CMDBs are critical for ITSM, but difficult to implement successfully
- A one-size-fits-all silver-bullet solution for CMDB design is very far away

- Patterns can serve as a toolset to aid in CMDB design
  - Common language for talking about CMDB data and information modeling
  - Building up a stock of reusable designs
- The three exemplary patterns presented…
  - aim to reduce complexity of the CMDB, while maintaining or enhancing utility;
  - do this either by reducing the number of CIs and relationships or by (to achieve a maintainable, easily visualized and understandable model)
  - Are general design patterns, but can serve as a basis for more specific patterns and CI templates (for common components, bits of infrastructure design etc.)

References

[1] *ISO/IEC 20000-1:2011 – Service Management System Requirements*, ISO/IEC, April 2011

[2] *ITIL Service Transition – 2011 Edition*, Cabinet Office, 2011

[3] Colville, R.J.: *Seven Steps to Select Configuration Item Data and Develop a CMDB Project That Delivers Business Value*, Gartner, January 2013

[4] Colville, R.J.: *CMDB or Configuration Database – Know the Difference*, Gartner, March 2006

[5] Adams, P. and Govekar, M.: *Hype Cycle for IT Operations Management 2013*, Gartner, July 2013

[6] England, R.: *ITIL's dead elephant: CMDB can't be done*, IT Skeptic (http://www.itskeptic.org/cmdb), June 2006

[7] Fowler, M.: *Analysis Patterns: Reusable Object Models*, Addison-Wesley, November 1996

[8] FitSM-5:2014: Guidance on the application and implementation of lightweight service management in federated IT infrastructures, FedSM project, http://www.fedsm.eu/fitsm, February 2014