A monitoring architecture supporting service management data composition

Vitalian A. Danciu
Munich Network Management Team
University of Munich
Oettingenstr. 67
D-80538 Munich, Germany
danciu@nm.ifi.lmu.de

Martin Sailer
Munich Network Management Team
University of Munich
Oettingenstr. 67
D-80538 Munich, Germany
sailer@nm.ifi.lmu.de

Abstract

Management disciplines beyond systems and network management require higher-order, enriched management information. To address this requirement, we outline a generic architecture that uses a specification language for the composition of management data to provide an integrated means of management information aggregation.

Keywords

Monitoring architecture, Service Information Specification Language, service management

1. Motivation

The monitoring of resources is a prerequisite for management. Existing management systems incorporate facilities to actively poll information from managed objects as well as listen for messages generated by them. Information distilled from this data is displayed to the administrator or triggers automated management actions. While this may be sufficient for some systems and network management scenarios, higher-order management disciplines like service management require aggregated and correlated information that cannot be obtained by isolated monitoring of single resources. Several specialized approaches have been devised so far. Clemm and Bansal approach autodiscovery of services by deducing service information from the configuration of network elements [1]. Keller and Kar propose to compute application service dependencies by analysing software package repositories[4, 3].

A transparent provision of refined data is necessary to enable a management view from a service perspective. A transport of the data can then be realised by means of an event propagation facility. We refer to this kind of events as *rich events*, since they transport enriched, aggregated data records, as opposed to common traps or system events that pertain to a narrow, specific application.

A service view may incorporate information from element, application and customer management views. However, the diversity of tools employed in these management disciplines impede the composition of monitoring data into records appropriate for service management. Furthermore, the problem of heterogeneity, yet unsolved in network management, is aggravated by the abstraction and aggregation necessary in service management: while

it would suffice that a network management system understands different protocols and data formats, a service management system must provide facilities to combine accquired data into a service view.
In this paper we propose a generic monitoring architecture for service management. It facilitates the creation of information records from raw monitoring data in a service specific manner. This allows differentiated monitoring of services that rely on a multitude of resources. By abstracting the heterogeneity of resources, a unified monitoring interface is within reach.
Our approach facilitates transparent creation of service management data that can be further utilized by a number of applications. They include process-based management where the resulting rich events serve as input parameters for management processes as well as policy architectures that require enriched data to trigger parametrised management actions.

2. Architecture

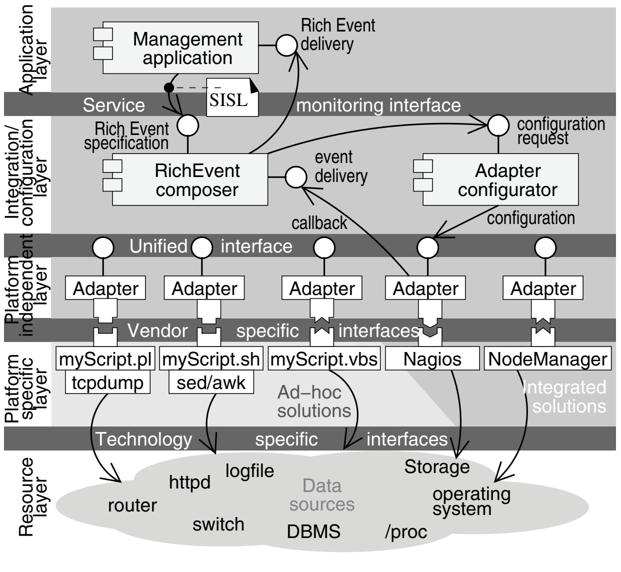

Figure 1: Generic monitoring architecture

Service monitoring relies on component, application and customer management to compose an integrated view of the services' state. The data gathered from system, network and application management tools needs to be combined in such a way that it reflects the service. To be able to perform the combination, dependencies between resources must be known. Also, service interdependencies must be taken into account. As in other management implementations (e.g. Internet management), a service management system should be able to poll data as necessary, but also to be notified of changes in the service by means of event transmission.
Since the same data may be relevant to several services, a monitoring architecture needs to allow flexible configuration. This includes temporal parameters (e.g. different polling intervals for the same data but for different services) as well as the creation of data clusters (as monitoring different services may require data from intersecting sets of sources). Since new services may require new sources of data, monitoring must be easily extensible while retaining the same interfaces for configuration and data transport. Figure 1 shows a layered architecture capable of satisfying the above requirements.

Resource layer and platform specific layer The lowest (resource) layer contains the data sources for monitoring. As suggested in the resource cloud, the data sources do not only differ in their set of attributes, but may include completely different types of objects. The multitude of management tools in the platform specific layer include integrated solutions, such as off-the-shelf management tools as well as 'homemade' scripts. In any typical scenario, operators complement integrated solutions with their own tools.

Platform independent layer The first step towards a homogeneous service view is a technology independent layer that hides the diversity of tools in the platform specific layer. In practice, this layer will consist of configurable adapters that use the platform specific layer according to a configuration given to them via a uniform interface.
Any new data source can be included by providing an adapter and a platform specific component (e.g. a management facility provided by a product vendor).

Integration and configuration layer Adapter configuration is performed centrally in the integration/configuration layer. This layer is dominated by two components whose functional scope includes configuration and data composition. Both access the unified interface provided by the platform independent layer. The *Adapter configurator* performs adapter setup in response to configuration requests. These requests originate at the *RichEvent composer* in accordance with the requirements of a service monitoring task. The specification of these tasks is formulated in the specification language outlined in Section 3.

Application layer After having been configured, the adapters relay data to the composer where it is aggregated into enriched events and made available to the management application. Though the architecture has been designed with service management in mind, it is suitable for policy based management approaches relying on rich event transport, as well as for Grid scenarios[2] , where information from a multitude of heterogeneous data sources must be aggregated.

```
 <aggregation name="webhosting_status">
   <infoitem>
<comment>Router must be up</comment>
     <source type="ping">
       <ipadr version="4">141.84.218.130</ipadr>
     </source>
     <interval unit="seconds">30</interval>
   </infoitem>
   <infoitem>
<comment>DNS host must be up</comment>
     <source type="ping">
       <ipadr version="4">192.168.1.56</ipadr>
       <interval unit="minutes">2</interval>
     </source>
   <infoitem>
<comment>Check service at http SAP</comment>
     <source type="httpget">
       <hostname>www</hostname>
       <port>80</port>
     </source>
     <interval unit="minutes">2</interval>
     <triggeronfail />
   </infoitem>
</aggregation>
```

Figure 2: SISL Example

3. A language for Specification of Service Management Information

The Service Information Specification Language (SISL) enables management applications to request aggregated information in a declarative manner, in that it allows combination of attribute sets with aggregation rules for management information. Thus, the dependencies between service information items are formulated explicitely and rich events purporting to service oriented attributes are made available.

SISL language elements by example Figure 2 shows a simple example of SISL for expressing the operational state of a web hosting service. Assuming that the service dependes on DNS and a router, these attributes (`infoitem`) are included in the `aggregation` element. The service is tested periodically at its access point and triggers an event/trap in case of failure. In the example, the monitored service is deemed to be healthy if all aggregated low level attributes show dependent services to be up. The elements representing information sources are typed; in the example this corresponds to the protocols used to access the low level attributes.

Acknowledgment

The authors wish to thank the members of the Munich Network Management (MNM) Team for helpful discussions and valuable comments on previous versions of this paper. The MNM Team directed by Prof. Dr. Heinz-Gerd Hegering is a group of researchers of the University of Munich, the Munich University of Technology, the University of the Federal Armed Forces Munich, and the Leibniz Supercomputing Center of the Bavarian Academy of Sciences. Its web–server is located at http://www.mnm-team.org.

References

[1] Alexander Clemm and Anil Bansal. Auto-Discovery at the Network and Service Management Layer. In *Integrated Network Management VII, Managing It All, IFIP/IEEE Eighth International Symposium on Integrated Network Management (IM 2003)*, volume 246, Colorado Springs, USA, March 2003. IFIP/IEEE, Kluwer Academic Publishers.

[2] Jeffrey Hollingsworth and Brian Tierney. Instrumentation and monitoring. In Ian Foster and Carl Kesselman, editors, *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 2003.

[3] G. Kar, A. Keller, and S.B. Calo. Managing Application Services over Service Provider Networks: Architecture and Dependency Analysis. In J. W. Hong and R. Weihmayer, editors, *Proceedings of the 7th IEEE/IFIP Network Operations and Management Symposium (NOMS'2000)*, pages 61–75, Honolulu, Hawaii, USA, April 2000.

[4] Alexander Keller and Gautam Kar. Dynamic dependencies in application service management. In Hamid R. Arabnia, editor, *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2000)*, Las Vegas, Nevada, USA, June 2000. CSREA Press.

Biography

Vitalian A. Danciu studied computer science at the Ludwig-Maximilian-University in Munich, Germany (LMU), where he received his diploma degree (M. Sc.). Since joining the MNM team in 2003, he is a Ph. D. candidate at LMU while working as a research and teaching assistant. His research interests include policy-based, process-oriented management, as well as management of mobile systems.

Martin Sailer studied computer science at the Munich University of Technology, Germany (TUM). He joined the MNM team after receiving his diploma degree (M. Sc.) in 2004 and is currently a Ph. D. candidate at LMU, where he works as a research and teaching assistant. His research interests focus on information modeling and service management.