

Improving Distributed Service Management Using Service Modeling Language (SML)

Robert Adams
System Technology Lab
Intel – Oregon, USA
robert.adams@intel.com

Ricardo Rivaldo¹
Information Technology
Department – QI College
Gravatá, Brazil
ricardo_rivaldo@gmail.com

Guilherme Germoglio,
Flávio Santos
HP Brazil
Porto Alegre, Brazil
germoglio@gmail.com,
flavio.barata@gmail.com

Yuan Chen,
Dejan S Milojicic
HP Labs
Palo Alto, USA
yuan.chen@hp.com,
dejan.milojicic@hp.com

Abstract—Automatic service and application deployment and management is becoming possible through the use of service and infrastructure discovery and policy systems. But using the infrastructure optimally requires intimate knowledge of the hardware and the interaction of its components in order to make optimal allocation of shared resources. This paper proposes an architecture where the hardware infrastructure not only makes operational parameters available (disk size, network bandwidth) but also presents to the service management components, relationships and constraints between the hardware components. We present an implementation which uses the Service Modeling Language, SML, to communicate this information and show how this architecture saves service management from knowing intimate knowledge of the hardware. This enhances optimal service deployment and management in a heterogeneous hardware environment and is a step toward autonomic computing.

Keywords—*model based management; Service Modeling Language (SML); hardware/software management integration*

I. INTRODUCTION

Modern data center service scheduling is becoming more automated. Technologies are being developed for discoverable service interfaces (SOAP [14], WSDL [4], UDDI [19]), service configuration (CDDL [2]), infrastructure configuration (CIM [5]) and common data representation (XML [20]). These technologies allow management programs to programmatically configure and allocate services onto an infrastructure with the goal of creating a ‘hands-off’, automatically managed system.

Current systems, though, tend to take a fairly simple and centralized view of the service and infrastructure. Such systems have a service manager that collects the requirements and configuration of the services and the resource infrastructure, calculates the correct deployment of the service onto the infrastructure and then causes the service to be deployed. The service manager then monitors the operation of the service and the infrastructure and reallocates resources and redeploys services to account for changes in resource availability, workload changes and other such dynamic changes.

A service manager, to calculate the mapping of services onto infrastructure resources, must have the knowledge of

¹ Work completed while employed at HP.

detailed resource interrelationships in order to do correct scheduling. Traditional solution is to implement this intimate resource interrelationship knowledge into the service manager. This implementation would be different for every piece of infrastructure and, as described above, is closely tied to the design of the hardware resources.

We propose an architecture where the underlying infrastructure communicates upward its dependencies and configuration. In order to calculate the correct (or optimal) deployment of the service onto its infrastructure this information is combined with its downward set of dependencies and configuration.

We present an implementation of this architecture using SML to add the additional constraint formation to the basic XML structure data. We also show the bottlenecks and opportunities for distribution and parallelization of the optimal mapping of services onto the infrastructure.

II. SERVICE MANAGEMENT

The class of workloads we manage in this paper are ones that consist of several services which are each built from multiple instances. The best state for a particular service is when its required number of instances is running. A service could be, for example, a tier in a multi-tier web service: one service is N instances of HTTP processor, another service is M instances of scripting engines and another service is Q instances of database engines. There exists a service manager responsible for creating the proper number of instances of each service.

We represent the structure of our system with Service Modeling Language (SML)[17][15]. SML extends XML schema validation with rules (Schematron [16]) which specify constraints between values that may appear in the instance document. An SML *model* consists of two classes of documents: *definition documents* which provide the description of how the instance documents should be structured and multiple *instance documents* which contain data for individual instances.

The definition documents contain constraints that are applied to the instance documents to verify they are valid. Constraints are captured in two ways: *Schemata* are constraints over the structure of data in a model and *Rules* are constraints authored

as Boolean expressions on top of the data in a model. SML uses a profile of XML Schema 1.0 as the schema language. The definition document rules are XPath expressions [3] which create assertions on the values in the instance documents.

There exists a *validation* operation which takes instance documents and validates them against the schema and rules of the definition document. If the assertion fails, a diagnostic message that is supplied by the author of the schema can be displayed.

We implement the SML validation as the front end of an ECA policy engine. The policy and selection operations of the service manager are created from input for the hardware and infrastructure resources as well as the overall, authored service policy. The details of this solution are described in later sections.

A. Hardware Characterization

One of the principal operations of a service manager is selecting the correct resources for the service. To make a reasonably optimal resource selection, knowledge of the hardware and software infrastructure is necessary.

The problem for a service manager designer is to include the logic for all these cases into the policy statements for service management. Because of the difficulty of the task, the common solution is to take an easy course and ignore the potential optimizations and code in a “fudge factor”. The problem with including any of the hardware information mentioned above is several fold: first the target hardware environment is usually heterogeneous meaning there are many different hardware servers from different manufactures; second, the service policy author usually does not have intimate knowledge of the internal architecture of the hardware; third, the service policy author usually does not have intimate knowledge of the interactions of hardware resources; and finally, even if the service manager captures all of the above, the hardware manufacturer will release a newer version of the hardware with different and subtler interactions.

To simplify this task in our architecture, the hardware and software infrastructure itself provides the equations for the interactions of the hardware resources. These equations for relationships are described as SML definition documents supplied by the hardware resources themselves. In this way, the service policy writer merely includes these equations into the service policy selecting, for instance, memory bandwidth capabilities of a hardware platform when placing a service instance.

III. IMPLEMENTATION

In this section we describe our implementation of a managed service which uses execution platform information in its management policy.

A. Infrastructure and Monitoring

The PlanetLab [12] test bed is a collection of several hundred computers (“nodes”) hosted at companies and universities around the world. Each of these computers can create virtual machines for an application on request. All of

these nodes are used by many researchers for distributed application research. Thus, it is easy to create an execution environment on any PlanetLab node, but one must contend with a computer which is being utilized by other services.

A service was added to each PlanetLab node to collect data on the node and all the running virtual machines. This information is collected by the *NodeMonitor* application which distributes the information on the Planetary Scale Event Propagation and Router (“PsEPR” pronounced “pepper”)[1]. This is a publish/subscribe event system which routes XML messages. A centralized application (*toRepos*) subscribes to these monitoring events and stores them in the repository. This is shown graphically in Figure 1.

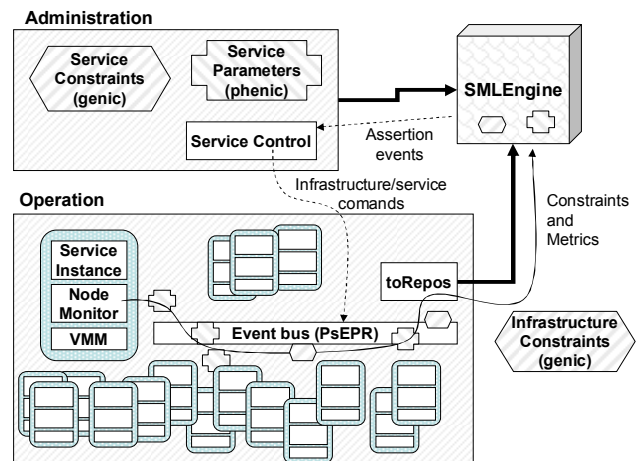


Figure 1: Block diagram showing the server instances running in multiple virtual machines, metrics information being sent over the event bus and being stored in the SML repository system. The dotted lines show the flow of validation and inferencing events to control the service instances and the infrastructure.

The definition documents for the hardware are also sent over PsEPR to the SML repository. Additionally, the service definition and instance XML documents are stored into the SML repository.

B. SML Validator Architecture and Design

Our service management implementation provides the following services: Storage, Validation, Inferencing, and Event notification. This is the central architectural component which collects all of the service specification documents (definition SML documents), the current system state (instance SML documents) and validates the current state to generate actions.

The *Storage Layer* stores both the definition and instance documents. This provides the place for the fusion of all of the policy definitions and the system state. SML does have syntax for external document references, but the referenced documents must also be pulled and be available to complete evaluation.

The *Validation and Inferencing Layers* operate on all of the data in the Storage Layer and output validation (schema) and inferencing (rules) results.

The *Publish-Subscribe Layer* is responsible for application subscription solicitations and notification of those applications when those models change.

The *Engine Core Layer* is the “glue” that makes all the layers work together, taking care of communication and management of shared resources and others specific implementation details.

The *Web Services Layer* is a web service stack compatible with WS-Addressing specification [21]. This layer is the front-end for the engine.

C. Service and Hardware Model

To control the services that are instantiated on the hardware resources, description of all the major components as SML definition documents are used. These fall into four groups: the Hardware Resources, the Service Model, the Service Parameters and Current Meta state. For each of these, there are definition documents that describe the potential values and assertions for that component and instance documents that hold the current parameter values.

For our implementation, the *hardware resources* are the PlanetLab nodes and the virtual machines that are created on them. The hardware capabilities are given by definition documents which describe the free resources on the individual nodes.

The base assertion of this paper is that the hardware resources themselves can supply its constraints and capabilities to the upper service manager. Since we did not have a test platform that allows for low level hardware characterization (power states, memory organization, etc) we implemented this feature by having the nodes supply the rules for their suitability for different types of services. For instance, a high load average of a node could make a node unsuitable for a compute intensive service.

We define four classes of nodes: COMPUTE1 which have available compute resources, NETWORK1 which have available network resources, RESPONSE1 which have better than average network response times and COMPUTE2 which have more compute resources than COMPUTE1.

The service instances specify which class they fit into and the hardware resources provide the definition of what the classes are computed. In this way, the service specification does not need to comprehend all of the details of the actual implementation of the capability.

SML, as defined, is not capable of being a general constraint language – the definition documents operate on the values of elements in instance documents and the expression in the definition documents only fire test assertions which do not generate new values for testing. This is the main problem we needed to solve to use SML for our application – the need to generate new, testable values. In our implementation, we overcame this problem by having the SML assertions generate new instance documents – the application that is calling the Validator receives the body of the fired assertions and, if that body, is an ‘event’, that event is placed into the repository This allows calculated values to become available for other assertions in other definition documents. Note that the generation of intermediate instance documents is not the same as the hardware devices performing the calculations internally and merely making the final value available for testing. Since

the test expression is available in the policy engine, it can include more variables than just the values in the local scope of the hardware. That is, the service pertinent input can be included in the calculation of the hardware capabilities. This makes our solution more general while using emerging and available standards.

IV. DEMONSTRATION

To demonstrate the operation of this architecture, we ran a synthetic service on nodes with various capabilities and loads. The characteristics of the resources (the assertions in the node definition documents) and the characteristics and state of the service are validated to drive service management actions such as creating new instances for the service.

Figure 2 shows the nodes which fall into each node class as collected over a period of time. The number of COMPUTE2 nodes is limited as seen in the graph. This covers a period of three days and the variance is due to the changing workload and network activity on every PlanetLab node (resource contention).

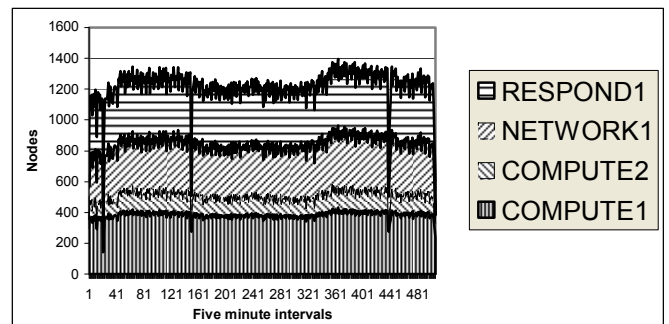


Figure 2 Number of nodes fitting the various capability descriptions.

Figure 3 captures some of the parameters for the service manager operation. The number of management actions is initially high and tapers off as the number of nodes is assigned. Notice, though, that the number of SML assertions and the number of generated events does not taper off as the allocated nodes reach equilibrium. This happens due to SML validation step, which has to be executed whether the output of that assertion is used or not. This demonstrates some of the scalability problems inherent with using SML.

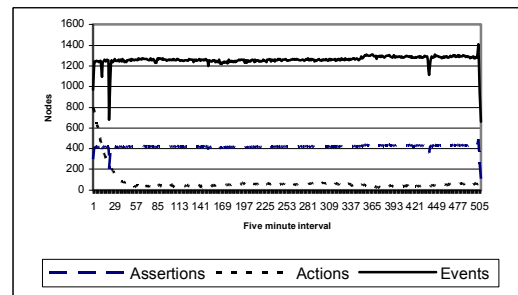


Figure 3: Count of fired SML assertions and the number of management actions taken as the characteristics of the nodes change.

V. RELATED WORK

Model-based management is not a new concept. As the services are becoming more complex, some efforts and alternatives are appearing to manage these environments [8][10][13][17]. The use of models to manage this complexity is emergent and discernible in different areas of computer science such as computer networks [6], distributed applications [7], or web services [9]. The purpose of this work is addressing service definitions and their relationship with network provisioning [6]. The services and policies are modeled using an XML-based language. However, de facto standards such XML Schema or XPath are not leveraged. Actually, the models in this language must be composed of specific primitives that must be understood by the language evaluation engine.

Compared with existing model based management, to our best knowledge, our work is among the first efforts toward a SML-model based management solution. The constructs and features provided by SML enable us to model and manage large and complex IT system with less effort. Inter-model reference support to model widely dispersed resources in a federated way. Further, the distinction of observed and desired states provides a natural way to manage distributed services, i.e., validate actual state against desired state and change the configuration through actions defined in policies if there is any derivation.

VI. CONCLUSIONS

This paper set out to explore two topics: use of SML for service management and improvements to that management through the hardware infrastructure reporting capabilities up using SML. The lessons learned from building such a system showed several difficulties in building such a system.

The SML specification has several characteristics that needed to be accommodated during the design phase. SML can test the values within an instance document but it cannot generate new values for later testing. Our implementation solved this by generating events which, in turn, were stored as new instance documents. Selection in SML is performed with XPath expressions. These expressions are evaluated within the SML Validator. This makes the expressions very hard to debug because 1) the evaluation is in a context that cannot be single stepped or easily observed, and 2) XPath 'fails' by quietly selecting nothing. The latter feature means that the slightest mis-coding will mean nothing happens (no selection means no assertions) with no easy way to discover why. Debugging is thus extremely difficult with no tools to help. SML also limits XPath expressions to XPath 1.0. This means no date/time operations. In our implementation, all times had to be kept as long integers (UNIX epoch times) that could be compared with regular arithmetic operations.

In addition to these design problems, many of the existing SML implementations are constructed so they do not operate in a real environment. In particular, 'compiling' implementations which generate XSLT or Python code from the definition documents makes real-time processing difficult and creates a complex debugging situation. Future SML implementations must take these real world requirements into consideration.

The concept of hardware characteristics and constraints generated and provided by the hardware infrastructure in order to reduce service management complexity was shown to be possible. Systems like SML which allow several sources of constraints to be merged together show promise and our future work will extend our implementation to a specific hardware and software system from the generalized environment described in this paper.

REFERENCES

- [1] Brett, P., et al., "A Shared Global Event Propagation System to Enable Next Generation Distributed Services", WORLDS'04: First Workshop on Real, Large Distributed Systems, December 2004.
- [2] CDDLM - Global Grid Forum CDDLM document, <<http://www.ggf.org/documents/GFD.51.pdf>>
- [3] Clark, J., DeRose, S. (editors), XML Path Language (XPath) Version 1.0 <http://www.w3.org/TR/xpath>
- [4] Christensen, E., Curbera, F., Meredith, G., Weerawarana, S., Web Services Description Language (WSDL) 1.1, W3C 15 March 2001, <<http://www.w3.org/TR/wsdl>>
- [5] Common Information Model ("CIM") Standard, Distributed Management Task Force, Inc, <http://www.dmtf.org/standards/cim/>
- [6] Copal, R., "Unifying Network Configuration and Service Assurance with a Service Modeling Language." IEEE NOMS, pp 711-725, '02.
- [7] Dearle, A. et al. "A Framework for Constraint-Based Deployment and Autonomic Management of Distributed Applications". ICAC, 04.
- [8] Eilam, T., Kalantar, M., Konstantinou, A., and Pacifici, G. "Model-based automation of service deployment in a constrained environment". Research report, IBM, 2004.
- [9] Foster, H., et al. "Model-based Verification of Web Service Compositions." IEEE ICASE, pp 152-161, 2003.
- [10] Garschhammer, M., Hauck, R., Hegering, H.-G., Kempter, B., Radisic, I., Rolle, H., Schmidt, H., Langer, A., and Nerb, M. Towards generic service management concepts - a service model based approach. In 2001 IEEE/IFIP International Symposium on Integrated Network Management Proceedings, pages 719-732, 2001.
- [11] Grid & Utility computing - <http://devresource.hp.com/drc/topics/utility_comp.jsp>
- [12] Peterson, L., et al, "A Blueprint for Introducing Disruptive Technology into the Internet", Proceedings of the First ACM Workshop on Hot Topics in Networking (HotNets), October 2002.
- [13] Rodosek, G.D.. A generic model for it services and service management. In IFIP/IEEE Eighth International Symposium on Integrated Network Management, 2003, pages 171-184, 2003.
- [14] SOAP - XML Protocol Working Group <http://www.w3.org/2000/xp/Group/>
- [15] Rivaldo, R., Germoglio, G., Santos, F., Chen, Y., Milojicic, D., Adams, R. SML Model-based Management. [Integrated Network Management 2007: 761-764](http://www.integrated-network-management.com/2007/761-764)
- [16] "Rule Based Validation - Schematron" - ISO/IEC FDIS 19757-3 , <http://www.schematron.com/iso/dsdl-3-fdis.pdf>
- [17] SML Specification - www.serviceml.org
- [18] Thompson, H.S., Beech, D., Maloney, M., and Nendelsohn, N (editors), XML Schema Recommendation, <http://www.w3.org/TR/xmlschema-1/>
- [19] UDDI - OASIS UDDI Specifications TC - Committee Specifications, <<http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm>>
- [20] XML - Extensible Markup Language <<http://www.w3.org/XML/>>
- [21] WS-Addressing - <http://www.w3.org/Submission/ws-addressing>