SML Model-based Management

Ricardo Rivaldo, Guilherme Germoglio, Flávio Santos
HP Brasil - Brazil
{first.lastname}@hp.com

Yuan Chen, Dejan S Milojicic
HP Labs - USA
{first.lastname}@hp.com

Robert Adams
Intel - USA
{first.lastname}@intel.com

Abstract — **Abstract models enable a clear separation between domain knowledge and application-specific details. In the system management arena, there are multiple implementations of model-based system management solutions, but until now, there was no industry-wide agreement on a common language or paradigm to enable interoperability. A new standard has been proposed to describe IT Services, the Service Modeling Language ("SML"). While SML enables interoperability, it still poses challenges in terms of scalable model stores, model validation, and use. This paper discusses an architecture for SML model management and validation and describe and evaluate a prototype implementation with a use case on system management of PlanetLab.**

I. Introduction

Model-based system management is not a new concept [8], [10],[11], [12], [15], but until now there was no industry-wide agreement on a common language to enable interoperability and model reuse. A new standard towards this objective is the Service Modeling Language (SML) [1]. While SML enables interoperability, it poses challenges in terms of designing model stores for performance and scalability, validating models, reasoning about actual versus desired state of a system and using the models.

This paper discusses the use of SML to manage a large scale distributed system (PlanetLab [2]) and evaluates an SML model store and validation. The paper is organized as follows. In Section II, we introduce the SML language. In Section III we demonstrate how SML is used by presenting a real use case (automated service management on PlanetLab). Section IV presents the architecture and design of the SML validator and model store. Section V shows the implementation of the validator. A scalability and performance evaluation of the prototype appears in Section VI. Finally, we compare our results with related work in Section VII and summarize the paper in Section VIII.

II. SML Language Overview

Service Modeling Language (SML) [1] is defined as an XML Schema derived modeling language built on profiles of existing W3C and ISO Schematron [16] standards.

A model in SML is realized as a set of interrelated XML documents which contain information about the parts of an IT service and constraints that each part must satisfy for the IT service to function properly. Constraints are captured in two ways: schemas and rules. *Schemas* are constraints over the structure of data in a model. SML uses a profile of XML Schema 1.0 as the schema language. SML also defines a set of extensions to XML Schema to support inter-document references. *Rules* are constraints authored as Boolean expressions on top of the data in a model. SML uses profiles of Schematron and XPath 1.0 for rules.

SML defines a *Document* as a well-formed XML 1.0 document and a *Model* as set of inter-related documents that describe an IT service. Each *model* consists of two disjoint subsets of documents – *Genic documents* and *Phenic documents*. A Genic Document is the subset of a document in a model which describes the rules and principles that govern the content of the model's documents. The specification defines two kinds of Genic documents: XML Schema documents that conform to SML's profile of XML Schema and rule documents that conform to SML's profile of Schematron. A Phenic Document is the subset of a document in a model that describes the form and structure of the modeled entities. *Model Validation* is defined as the process of verifying that all documents in a model are valid with respect to the model's Genic documents. Validation of a model is done by a *Model Validator*, which is an embodiment capable of performing model validation.

III. Use case: automated service management of PlanetLab

To demonstrate the above described features of SML, a dynamically managed service use case has been built and tested. The testbed environment used (PlanetLab) has several features that fit requirements of "utility computing"—dynamic allocation of virtual machines, geographic distribution, and dynamic workload on each server. A system where resources are dynamically allocated as required by a service and/or a service level agreement encourages the use of a model-based system. The SML Model Validator is used to test whether a version of the infrastructure and service operation conforms to the rules and constraints of the model. The Model Validator output feeds a service management module to perform actions to change the service operation so that it conforms to the models. The action module and the SML Model Validator together comprise policy based service management.

IV. Architecture and Design

The software is designed to provide users with a set of services to work with SML models and implements functionality required by an application using SML models. The architecture is presented in Figure 1.

Validation and Inferencing Layers perform what is defined as model validation. The SML language has two aspects: syntactic and semantic. The syntactic aspect of the language expresses the model's data structure. The semantic aspect of models is a knowledge representation of IT Services expressed as Schematron rules. This knowledge can be used to verify differences between the actual and desired service state, i.e., to check semantic compliance with the application domain. Validation is the process of verifying that the model and instance data syntax and semantic are correct.

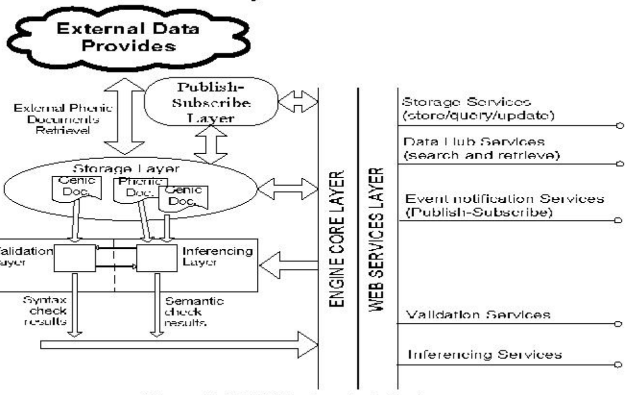

Figure 1: SML Engine Architecture.

The service will return the syntax and semantic results to the caller. These results may be error messages if the document is not syntactically correct or the text output produced by the Schematron rules on the corresponding Genic document. The SML Engine will treat inter-document references and take into account where Phenic documents are located, i.e., locally or external to the repository.

Storage Layer. Both Phenic and Genic Documents can be stored, updated, queried and retrieved. The storage layer will find and retrieve not only locally stored Phenic documents, but also external ones. In the storage process, an application can ask for a Validation and/or Inferencing action to guarantee that the stored model is syntactically and/or semantically valid.

Publish-Subscribe Layer is responsible for handling applications subscription requests on SML Models change (Phenic and Genic Documents) and for notifying those applications when those models change.

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.

Web Services Layer, as the name suggests, is a web service stack compatible with WS-Addressing specification. This layer is the front-end for the engine.

V. Implementation

Our implementation uses modified open source tools: XMLBeans [17] as the interface to the XML Schema validator built from Scimitar (a component of Amara XML toolkit [18]) as an ISO compliant Schematron validator. We extended Scimitar to develop an SML Identity Constraints validator and an XML library to handle the XML hierarchical structure and to treat inter-document references.

SML model validator is composed of four main layers:

1. *Documents Publisher* accepts the set of files with their respective aliases and creates a map where, given an alias to a file, its URI can be retrieved. This entity eases the inter-document references resolution.
2. *XML Schema validator* validates the entire model against the SML and W3C XML Schemas.
3. *Schematron validator* includes: a) Schematron Extractor: Transforms the model schema in order to get all the Schematron rules in a unique XML file. b) Scimitar Parser: Gets the Schematron Extractor output and generates a Python validation file. c) Rules Evaluator: Executes the Python file and returns a significant code (zero if everything valid and another number otherwise).
4. *Identity Constraints validator* includes: a) Constraints Extractor: Creates a set of constraints to be analyzed against the date files. b) Date Selector: Uses the constraints definitions, such as 'selector' and 'field' elements, to select the data to be analyzed. c) Constraints Analyzer: Analyzes each constraint (key, unique and keyref) and returns a significant code.

PlanetLab SML model represents a collection of several hundred computers ("nodes"), each able to create virtual machines on request. A service runs on each PlanetLab node which collects data on the node and on all running virtual machines. This information is collected by the NodeMonitor application which places it on the Planetary Scale Event Propagation and Router, PsEPR (pronounced "pepper"). PsEPR 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. The PlanetLab structure model is a straight forward hierarchical organization of the nodes within sites. The model for the metrics [3] includes per-node and per-virtual machine measurements. The service instances also generate metric information on operations which are part of the service model. Part of the service model is the parameters that control the service and the assertions around these. In particular, the number of instances of the service to run is an important parameter.

The SML model for PlanetLab consists of: a) sites hosting nodes; b) the Nodes: which are hosting virtual machines; c) the Virtual Machines: where the applications are executed; d) the Services: describe the required resource and conditions for a service.

The Nodes and Virtual Machines levels have the metrics such as: CPU usage, bandwidth available, bandwidth usage, and virtual machines running on a node. For the relationships between nodes, virtual machines and service, Schematron assertions infer the state for at least the following conditions: Insufficient Nodes, Too Many Nodes, Failed Nodes Exist, Timeout Allocating, and Timeout Failed. The SML Validator checks those conditions each time the model's Phenic documents (the data) is updated and reports to the Service Control the inference process results. With those results, the Service Control can take the appropriate actions to correct the problems detected, if any. Those are initial, failure tests of node management. We will add tests for the health of the service instances.

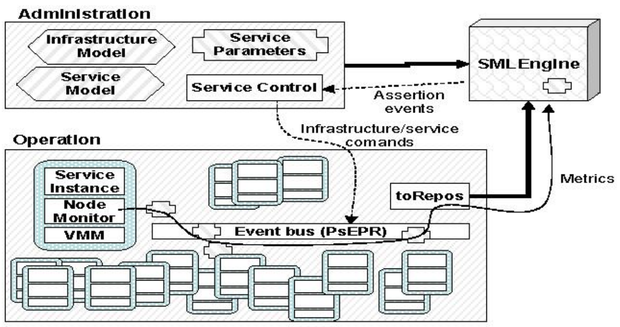

Figure 2: Server instances runs in multiple VMs, metrics information is sent over the event bus and stored in the SML repository. The dotted lines show the validation events to control service instances and infrastructure.

VI. Scalability and Performance Evaluation

A. Scalability of Validator. To evaluate the scalability of the validator we choose SML specific metrics.

*Inter-document Reference f*rom one document to another is introduced by SML. Our validator test shows that Inter-document reference performance is linear.

Phenic Documents: Figure 3 demonstrates the validation time as the function of the number of Phenic documents (instance data files) in a model.

Schematron Rules: The SML models describe rules based on Schematron. To measure the models of different size, the Figure 4 presents the validation time as a function of the number of rules.

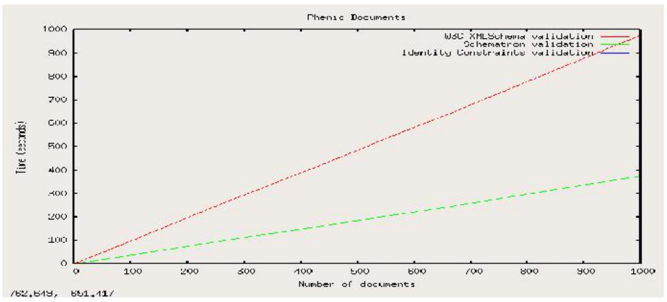

Figure 3: When a new Phenic document is added in a model, the entire validation time is the previous plus the new document validation time.

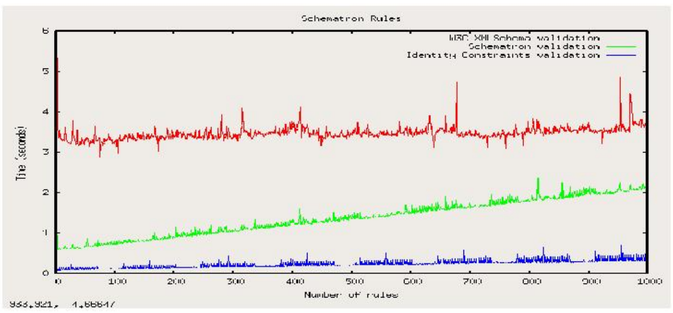

Figure 4: The phase with more resolution impact is the Schematron validation, since it has to evaluate the Schematron rules.

SML Identity Constraints. SML specifies three identity constraints (key, unique and keyref), however, we present experiments only for key constraint. We created three sets of files. The first uses two fields as a key, the second set uses ten fields, and the last set uses twenty. Figure 5 demonstrates validation time as function of the number of identity constraints for each set of files (simple, medium and complex). Figure 5 shows validation time as a function of the number of key constraints.

Observing the experiments results we can conclude that the XML Schema validation time only increases when the size of Phenic documents in a model increases due to the number or the length of the files. The figures 4, 5 and 6 show increasing of the validation time due to:

- Number of inter-document references: In order to increase the number of references, they needed to be defined in the Phenic documents, increasing the document length.
- Number of Phenic documents: The experiment itself increases the number of Phenic documents in the model.
- Number of Schematron rules: The number of rules evaluation increases when there are more elements in the Phenic documents where the rules must be applied. This increases the length of the documents.

The Schematron validation extracts the Schematron definitions from the Genic documents and applies them to the Phenic documents. Therefore, its validation time increases as these documents increase and this is true in all the experiments. The SML Identity Constraints validation time increases when the number of fields which compounds the constraints, the number of its occurrence in the Phenic documents and the Genic document size increases.

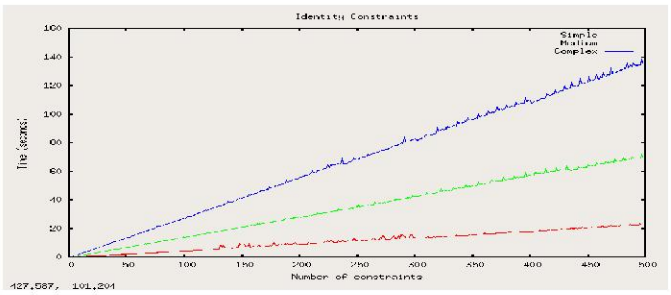

Figure 5: The linear behavior on constraints evaluation.

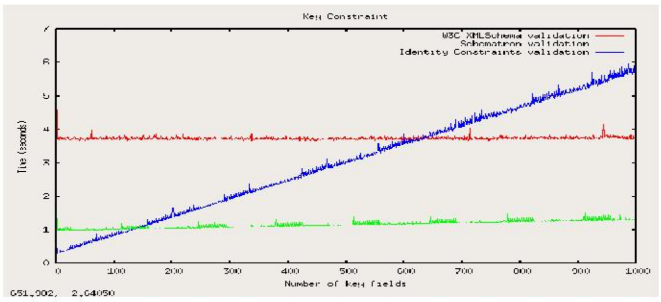

Figure 6: Depending on the number of fields in a key definition, the Identity Constraints validation phase can take more time than the others

A. Performance of automated service management on PlanetLab

A naive implementation of the policy engine would poll the Validator on every instance change. A more reasonable implementation is to evaluate the instance information at the response granularity required for service configuration changes. It becomes obvious that even continuous validation (which generates events when assertions are incorrect) can be quickly overwhelmed by the computational requirements of handling the number of instance document changes generated by even simple systems.

VII. Related work

The more complex services become, the more laborious their management is. Considering this, modeling has been the natural answer for reducing the management complexity. Many efforts have been made towards this [12], [15][13], [10]. Using models for management comprise different areas such as computer networks [7], distributed applications [9], or web services [11]. A lot of research was conducted on ECA systems and data repositories, and languages to specify actions for certain conditions and events [6][4]. Some of them are also very similar to SML [14], but none of them leverage standards, such as XML, XML Schema, Schematron, and XPath.

VIII. Summary and Future Steps

In this paper we proposed the architecture of an SML Repository Engine. The target application was a PlanetLab test bed to control Services running on dynamically allocated virtual machines. Our focus was implementing the model Storage and Validator and test rather then on the simulated models. We discussed the achieved validator performance. The functionality we provide could be useful to any application that works with SML Models. To make a completely application and programming language neutral tool we used web-services for communication. In the future work, we intend to address the following questions:

1) XML Storage and Performance

Experiments based on existing PlanetLab data estimate a 7x increment on the data size when converting a relational table to SML: each node outputs metrics once every 5 minutes. 37 days of collected data results in the MySQL table occupying 155MB and the corresponding XML is 736MB. This is a big concern to a running system which requires good performance of the XML parsing, store, retrieve, and validation. To address this, we plan to add a native XML Database and an event-driven notification engine.

2) Versioning on SML models

The main question is how to work with models that change over the time and keep them compatible with historical data.

3) SML Validator Reference Implementation

We are using an existing XML Schema validator associated with a Schematron validator [18], which works by generating Python code from the Schematron rules. This was good enough for a research system, but production systems require implementation of the overall SML validation process (including a Schematron processor) without intermediary code generation. This will permit us to work on specific optimizations for large scale SML models.

4) Large Scale Inter-document Performance

Besides the performance on model validation and storage, there is a need to investigate how the system will perform with SML inter-document references on large scale models.

Acknowledgment

We are indebted to Phil Prasek and William Vambenepe for their feedback.

References

[1] SML Specification - www.serviceml.org

[2] L. Peterson, et al., "A Blueprint for Introducing Disruptive Technology into the Internet", 2002

[3] Park, K. and Pai, V.S.2006. “CoMon:a mostly-scalable monitoring system for PlanetLab”. SIGOPS Oper.Syst. Rev.40, 1 (Jan. 2006), 65-74

[4] J. Bailey, A. Poulovassilis, and P. Wood. An event-condition-action language for xml, 2002.

[5] A. Bonifati, S. Ceri, and S. Paraboschi. Active rules for xml: A new paradigm for e-services. The International Journal on Very Large Data Bases, 10(1):39–47, August 2001.

[6] A. Bonifati, S. Ceri, and S. Paraboschi. Pushing reactive services to xml repositories using active rules. In A. Press, editor, Proceedings of the 10th International World Wide Web Conference, pages 633–641, 2001.

[7] R. Copal. Unifying network configuration and service assurance with a service modeling language. In 2002 IEEE/IFIP Network Operations and Management Symposium, pages 711–725, 2002.

[8] J. E. L. de Vergara et al., Semantic management: advantages of using an ontology-based management information metamodel, 2002.

[9] A. Dearle, G. Kirby, and A. McCarthy. A framework for constraint-based deployment and autonomic management of distributed applications. International Conference on Autonomic Computing, 2004.

[10] T. Eilam et. al. Model-based automation of service deployment in a constrained environment. Research report, IBM, 2004.

[11] H. Foster, S. Uchitel, J. Magee, and J. Kramer. Model-based verification of web service compositions. In 18th IEEE International Conference on Automated Software Engineering, pages 152–161, 2003.

[12] M. Garschhammer et al., Towards generic service management concepts - a service model based approach. International Symposium on Integrated Network Management, pages 719–732, 2001.

[13] G. D. Rodosek. A generic model for it services and service management., 2003.

[14] M. Schrefl and M. Bernauer. Active xml schemas. In Revised Papers from the HUMACS, DASWIS, ECOMO, and DAMA on ER 2001 Workshops, pages 363–376, London, UK, 2002. Springer-Verlag.

[15] B. C. Williams and P. P. Nayak. A model-based approach to reactive self-configuring systems, 1996.

[16] Schematron- http://www.schematron.com/iso/dsdl-3-fdis.pdf

[17] XMLBeans - http://xmlbeans.apache.org/

[18] Amara XML Toolkit - http://uche.ogbuji.net/tech/4suite/amara/