Dynamic Model-based Management of a Service-Oriented Infrastructure

Félix Cuadrado, Rodrigo García-Carmona, Juan C. Dueñas
Departamento de Ingeniería de Sistemas Telemáticos
ETSI Telecomunicación – Universidad Politécnica de Madrid
Madrid, Spain
{fcuadrado, rodrigo, jcduenas}@dit.upm.es

Abstract- **Models are an effective tool for systems and software design. They allow software architects to abstract from the non-relevant details. Those qualities are also useful for the technical management of networks, systems and software, such as those that compose service oriented architectures. Models can provide a set of well-defined abstractions over the distributed heterogeneous service infrastructure that enable its automated management. We propose to use the managed system as a source of dynamically generated runtime models, and decompose management processes into a composition of model transformations. We have created an autonomic service deployment and configuration architecture that obtains, analyzes, and transforms system models to apply the required actions, while being oblivious to the low-level details. An instrumentation layer automatically builds these models and interprets the planned management actions to the system. We illustrate these concepts with a distributed service update operation.**

Keywords- Model-Based Management; Runtime Models; Service-Oriented Architecture; Service Configuration; Service Deployment.

I. INTRODUCTION

In engineering, models are abstractions or conceptual objects used in the creation of a system. Model Driven Engineering is a methodology based on the systematic use of models in software development processes [1]. It aims to alleviate the complexity of current IT applications and infrastructure platforms and express domain concepts effectively. In their usage as software and hardware development tools, models become blueprints to build systems. The initial models are increasingly refined and enhanced by means of transformations, until the final system (the code) emerges [2]. The final model should represent the system "as it must be". Reality shows us that this source code must be frequently refined or modified by hand: code is changed throughout its lifetime and it is seldom synchronized with the original design models. In systems engineering the same situation happens; network and system design models, once defined and deployed, are disconnected from the real situation. This problem is exacerbated after release, during their operation time. Each change further decouples the models from the reality, invalidating the model for runtime reasoning.

Because of these limitations, we propose to use dynamic models; whose grammar (the metamodel) is defined beforehand, during the design phase, but whose information or data is obtained in runtime. These models are a key tool for system management, as they allow reasoning over the system "as it is", and conform with the system "as it must be". Thus, it is possible to automate the decision making activities related to system management, and to perform them based on these models depicting the actual state. We have combined static and dynamic models of running systems, in order to provide automated deployment and configuration for service oriented architectures.

The article is structured as follows. Next section provides an overview on the context of application, identifying the main requirements and concerns. Section 3 first discusses the suitability of the main existing information model standards for autonomic service management, and after that presents our proposed information model abstractions. The next section presents a management architecture that builds on the dynamic model approach. Section 5 expands the main ideas of our proposal through the explanation of a case study. Finally, the article is closed with the main conclusions and potential lines of future work derived from the presented results.

II. CONTEXT OF APPLICATION

The increased importance of IT infrastructure has led to significant investments in infrastructure, which must be amortized over long periods of time. However, systems evolve rapidly, rendering purchased units as legacy technology before their lifetime has been completed. On top of that, it is necessary to upgrade applications and enterprise services, and acquire new equipment, in order to continuously improve process efficiency to gain a competitive advantage. Thus, systems are composed by not only legacy systems, mainframes, or databases, but also Java Enterprise Edition (JEE) application servers, or Business Rule Manager (BRM) systems. The resulting enterprise infrastructure is a complex heterogeneous distributed system, composed by dozens of different servers and application containers, deployed over hardware machines interconnected through complex network distributions, containing firewalls, virtual private networks and other access restriction and security mechanisms.

Functional interoperability between all the components of the IT infrastructure is usually achieved by adopting a higher-level integration layer, which is based on Service Oriented Architecture and Business Process Management

(SOA/BPM). This way, each artifact of the system is presented as a service, hiding its implementation details and providing a uniform high-level view. Services are published in directories and connected through an Enterprise Service Bus (ESB), where additional non-functional capabilities can be added to the communications, like logging, or data transformation. On top of that, BPM technologies, such as BPEL (Business Process Execution Language) engines, orchestrate the activities, bridging the gap between the IT infrastructure and the business processes.

The SOA/BPM abstraction maximizes the use of the existing IT infrastructure, but managers still have to cope with the underlying heterogeneity and complexity, while supporting three key business requirements: controlling operation costs, warranting the quality of services and handling the evolution of the services and the infrastructure.

Traditional management processes are identified with human operation over a management administration console. Monitoring information and events are collected and aggregated into the console, and the administrator invokes specific operations on the environment based on the identified objectives and the collected information. Operations are executed in scripts, containing the exact set of machine-specific instructions for achieving a specific task. Because of that, scripts lack reusability and suffer from the increased complexity, distribution and heterogeneity of current IT systems.

The limitations of this approach become more evident as the complexity and heterogeneity of the managed systems keep growing. Changes to the environment impact the complete management process, as the configuration and workflows must be manually adapted to the specifics of the environment. Management operations are manually created and composed, requiring specialized knowledge from the administration experts, and can hardly be reused. On top of that, a runtime system configuration has dependencies between heterogeneous artifacts, propagating the impact of any change or error throughout the whole system.

There is clearly a necessity of reducing complexity, and lessening human intervention by automating parts of the management processes. These problems can be alleviated by using models; Model Driven Management [3] is a new approach for management, where models allow the abstraction from the complexity of the environment. This approach has numerous advantages over others, thanks to the greater expressivity of models [4]. We build on this approach, and propose the need to handle both static models containing the definitions of the developed services, and dynamic models that contain information directly obtained from the running systems (monitoring information); these models must be related and transformed in order to generate control actions that will be themselves represented by models, able to be applied on the managed systems by the proper agents. To verify this approach we have built a deployment and configuration system, which operates on models for characterizing the services, the runtime environment and the operations on a service oriented architecture.

III. MANAGEMENT INFORMATION MODEL

The management of networked systems is defined by [5] as all the measures necessary to ensure the efficient and effective operation of a system and its resources, based on the organization goals. However, the scope of distributed management greatly varies ranging from network, resource, application, service and business concepts. Regardless of the scope, every management system requires an information model that provides a homogeneous view of the managed elements. While there are successful proofs of concept of the implementation of autonomic managers using ad-hoc models and ontologies [6], the lack of alignment to existing standards and models greatly complicates its applicability to general cases. The information model must include all the relevant information for the management operations, the elements, its characteristics and relationships, while at the same time it must be flexible enough to adapt to heterogeneous environments and be as compatible as possible with the existing information modeling standards.

A. Information Modeling Standards

Several standards have been defined for modeling the relevant management information of a distributed system. Alternatives range from mature standards from the network management domain to emerging initiatives from the Internet domain. Here we present a brief overview on the most relevant ones, showing how they support service management.

MOWS (Management Of Web Services) [7] is an OASIS standard that defines how to represent Web Services as manageable resources. MOWS is part of the Web Services Distributed Management (WSDM), a set of standards from OASIS devoted to the management of IT distributed systems using Web Services technology.

The Common Information Model (CIM) [8] is an information model standardized by The Distributed Management Task Force (DMTF) industrial association. CIM is an object-oriented model for describing overall management information in a networked enterprise environment. CIM is specified as a set of UML models and complementary MOF (Managed Object Format) files, textual files expanding the semantics of the defined elements. CIM is a modular, extensible standard; it models a very broad set of elements including databases, networks, user preferences, and applications among others. Internally, CIM is divided into a core model, defining the basic elements, and additional models extending from the base elements with additional details on one specific area (application, network, computer are some examples of profiles).

The OMG Deployment and Configuration Model [9] provides a simple and flexible model for representing deployment and configuration operations over a distributed target. The target environment is called a domain in its terminology, and is described by an object-oriented information model. The base elements of the domain model are resources, which are named entities classified into one or more types. Resource instances model physical artifacts, mainly: nodes, bridges and links.

There are two common characteristics of information models that cover systems and services: the use of object-oriented abstractions and the resource concept as the essential unit of management.

B. Proposed Information Model

After evaluating the existing information modeling standards, we have defined a set of modeling abstractions that try to effectively capture the relevant information of a heterogeneous distributed environment. The metamodels build upon the common ground shared by the standards, with D&C being the base reference because of its flexible nature.

Our service deployment and configuration architecture is supported by three metamodels, governing the static definitions of services, the runtime description of the environment and the planned management operations to the runtime environment. These metamodels complement each other, so they completely cover the required information for the management of enterprise services. The metamodels have been defined in EMOF (Essential MOF, a subset of the OMG's Management Object Facility MDA language)

The three metamodels share a core concept that is the base of both static and dynamic abstractions; the resources. A resource is a manageable element. Resources are characterized with a name, a version identifier and a set of properties. The resource definition is complemented by a type field, which establishes a resource taxonomy, inherently classifying the basic assets of the infrastructure environment (ranging from services to containers). This allows management systems to define actuators and policies that automatically apply to the matching elements. The concept has been taken from OMG D&C and expanded with versioning information for software resources.

The software metamodel provides a software architect-friendly abstraction for modeling software components, known as deployment units in our terminology. Units model their provided services and external requirements using the resource concept. Finally, deployment units can also specify environment constraints needed for correct performance (such as existing system services, available disk space or minimum amount of RAM memory) as mandatory resources of the runtime domain. Examples of typical deployment units include Java EE WAR and EAR files, Database DDL (Data Definition Language) and DML (Data Modification Language) scripts or BPEL (Business Process Execution Language) process definitions. Instances of this metamodel are considered static for the sake of configuration and deployment, because they are inputs to the management processes coming from the development infrastructure. The software model combines aspects from CIM Application model with the resource concepts. A previous version of this model is presented in [10].

Once components and services are deployed, they become runtime entities that are part of the environment, with a state and a specific configuration. The runtime metamodel defines the topology and the configuration of the managed environment. An environment is composed of a set of distributed nodes, interconnected through a network. Node configuration information is also modeled as resources and properties, whereas the network structure is defined by D&C's interconnect and bridge elements. On top of that, nodes host containers, where components and services are deployed. Examples of containers include an application server, a business process manager or a database. Runtime units are the main elements of the metamodel. They represent deployed units, providing status and runtime configuration information. Runtime models are generated on the fly, as they represent the environment for management purposes. Therefore, we need agents to extract the information and populate the models. The combination of both metamodels keeps the traceability between the static view of software development and the dynamic view of service management.

Finally, we have defined the management operations that can be applied over the runtime elements (resources, containers, and deployed units) as the plan metamodel. Because of the dependencies and inter relationships existing in a distributed system, operations cannot be executed individually and must be aggregated in plans. A plan is a collection of activities (such as unit deployment, component activation, or resource configuration) which must be executed over the environment to achieve a management objective (e.g., release a new version of the client user application). The activities are included in a directed graph, ensuring a correct execution order while allowing at the same time parallelization of non-dependant tasks. This definition constitutes a management DSL (Domain Specific language). Activities refer to the concepts described in the software and runtime metamodels (such as deployment units, resources, configuration values and containers). This abstraction allows a management system to dynamically create plans for achieving a specific goal through model reasoning, instead of the traditional mechanism of a system administrator manually defining change workflows.

IV. SERVICE MANAGEMENT ARCHITECTURE

In an enterprise environment the availability of the complete runtime information is vital for an effective management. The impact of changes spreads over the environment, due to the nature of heavily distributed systems. Additionally, the heterogeneity of the elements complicates coordinated efforts [11]. Because of those factors, our management functions are centralized for the complete environment, and reason over generic models abstracting from the specific system details.

Fig. 1 provides a high-level overview of our system. On the left-hand side we can see the main blocks of the management system (such as plan generation, Service-Level Agreement monitoring, or environment asset management). The management functions provided by the architecture range from static information processing (e.g., unit description storage, configuration and policy definition) to dynamic reasoning (e.g., environment monitoring, and plan model generation). The inputs and outputs of these components are model instances of the previously described metamodels, which isolates the functional components from the specifics of the managed environment. Additionally, the architecture implements a closed control loop. The

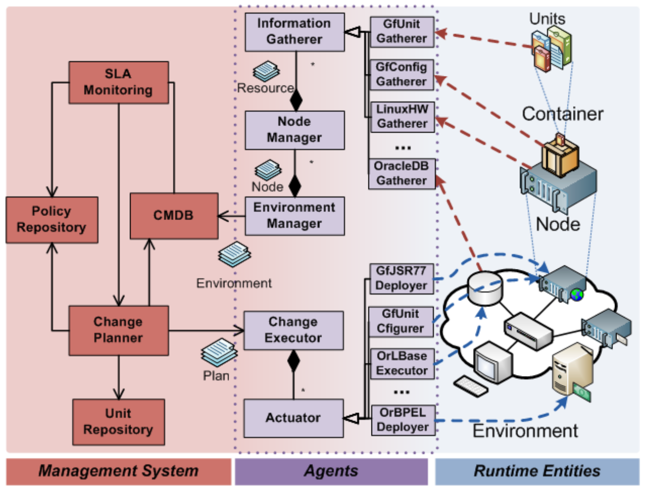


Figure 1 High-level Architecture View

configuration and status are continuously monitored by the agents, and transformed into runtime models. This information is processed by the systems, also taking into account high-level objectives and policies. The results of these processes are the deployment and configuration change plans, which are interpreted by the agents.

The physical managed elements are pictured on the right hand side. The impedance mismatch is addressed by the agent layer (in the central part of the figure), which transparently instruments the runtime elements and builds a coherent model with all that information. Communication goes both ways. The agents produce a model abstraction of the physical environment for the management blocks. At the same time, they apply plan models to the runtime infrastructure.

Depending on enterprise policies, business requirements and architecture decisions the composition of a runtime environment differs significantly. The instrumentation layer must automatically adapt to different physical configurations, without the need of manually defining the topology of the specific environment. Because of that, the agents' architecture follows a layered, extensible model.

The top-level agent is the *Environment Manager*, which acts as the contact point between the environment and the management system. It provides model descriptions of the environment and orchestrates the execution of plan models. Internally, it uses the DNS-SD protocol for automatic agent discovery. This way, the environment topology is dynamically built, although it is also possible to manually populate it to reflect runtime systems behind restrictive firewalls. The discovered agents are the *Node Managers*, which govern the resources, configuration and services available at node level. Information about hardware and software resources, containers and application is captured to the environment model by the Information Gatherer agents. These components instrument a specific aspect of the node, such as hardware information, JEE application server resources or service configuration. A *Node Manager* collects the gathered models for providing to the *Environment Manager* a complete characterization of the node. The operation infrastructure is composed of the *Change Executor*, which receives execution orders for change plans to the environment, and multiple *Actuators*, which apply the activities to the physical elements. These elements are aggregated similarly to *Gatherers*; each one is capable of applying one or more operations (i.e., configure container, install deployment units) on some parts of the environment. We can see how the base instrumentation elements are generic, and only the endpoint *Gatherers* and *Actuators* mediate between the runtime models and the runtime system. These agents perform the key transformations for dynamic model management, as they convert Specific Models from each management interface to our Platform Independent Model (the runtime metamodel), and interpret our plan DSL elements as invocations to vendor-specific commands.

V. CASE STUDY

We will present an industrial case study to illustrate how the elements of the management system collaborate for obtaining and applying configuration and deployment changes. The scenario has been extracted from the ITECBAN project, a Spanish Research project from the CENIT program whose objective is to develop a SOA-based core banking solution. A banking organization internally uses a credit grant service. It combines the use of inference engines and the input of human experts to provide a response to the requestor. On a technical view, the service is composed of clustered JEE applications, BPEL processes, Business Rule definitions and database information. These components are deployed over a distributed environment, and communicate through Web Services. After a user reports a service fault, the incidence is escalated to the development team, which releases an updated version of the faulty service, and a change request is issued to provision these modifications to the runtime environment.

Service update is a complex process, involving installation, life-cycle control and configuration of several, inter-dependent artifacts. Its correct execution requires retrieving and processing information about the current environment state, and the logical changes to the affected services. We will describe how this scenario is supported by collecting the state from the physical elements, obtaining a change plan from the static and dynamic models, and applying the changes to the environment.

The initial step is to obtain an updated snapshot of the environment, which will be represented by an updated runtime model. At this point, each *Information Gatherer* connects to the management interfaces of a monitored part of the infrastructure, obtains its current state and transforms that information to our modeling abstractions. As an example, Glassfish (the reference implementation of the JEE standard) *Gatherers* access the remote JMX Server, query and retrieve the relevant JSR 77 MBeans and transform that information into container, unit, resource and property elements. An analog process is applied to the rest of instrumented infrastructure, such as Oracle configuration and information, and Hyperic HQ inventory model instances. The *Node Managers* and *Environment Manager* aggregate that

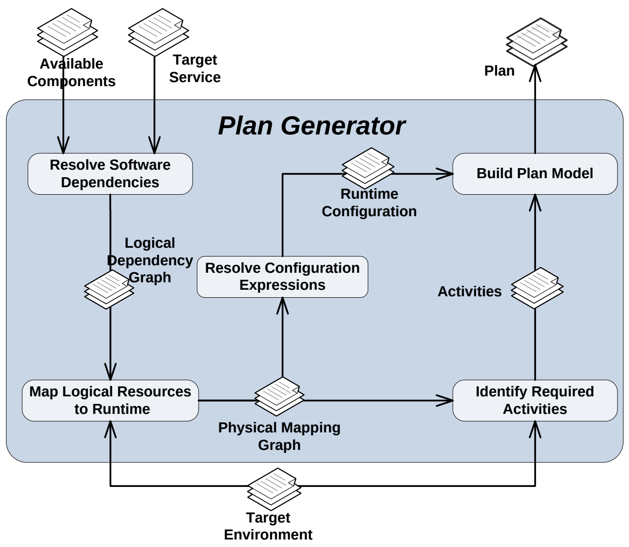


Figure 2 Plan Generation Activity Diagram

information and provide an updated environment model to the management system.

Once the updated model has been generated, the management components analyze the objectives and current status, and produce a change plan that will contain the required changes for updating the selected service. Fig. 2 shows the general update change execution flow. Functional elements operate with model instances, both as input parameters and as execution results.

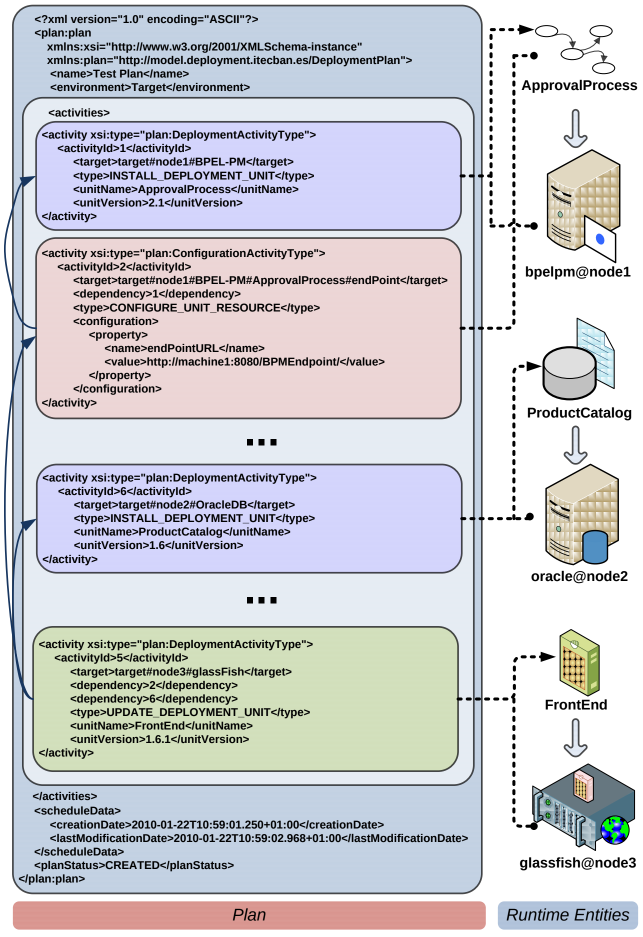
The process starts at the *Resolve Artifact Dependencies* step. The plan generator takes the descriptor of the service to be updated, along with the models of all the available components, and obtains a model graph representing the logical dependencies of the provider. Both current and planned versions are analyzed, in order to estimate the impact of the operation to the rest of the system. In case the update operation can be safely executed, the workflow goes on in order to generate the update plan.

After the logical dependencies have been calculated, the plan generator retrieves the updated environment model, and combines it with the logical graph in order to *Map the Logical Resources to Runtime elements*, designating a container for each logical unit (deployable artefact). Whenever more than one container is suitable for a given component, a decision is made either by an administrator or a distribution policy.

After obtaining the physical component distribution, the process will *Identify the Required Activities* for reaching the desired state, as well as the restrictions in their execution order. The state of the runtime environment is also taken into account in order to only generate the mandatory activities, avoiding redundant actions (e.g., if a deployment unit is already running on the selected container, an installation activity won't be generated for it). In parallel to activity generation, the executor will *Resolve the Configuration Expressions* of the graph components, automatically calculating the required configuration changes to services and environment in order to work correctly after the update operation.

Finally, by combining the configuration parameters and the plan activities, the change executor *Builds the Plan Model*. The complete plan is composed by 36 activities that represent the management operations. Each activity defines the operation to be performed, the element of the environment (such as a container resource, or a runtime unit) where it will be applied, and the operation arguments. Fig. 3 shows a fragment of the resulting plan, composed by four activities represented in XML syntax: two installation activities, one update activity and one configuration activity. The plan also contains the mandatory dependencies between them, to assure its correct execution.

Once the plan has been created, it is processed by the *Change Executor*. First, it identifies which *Actuator* will perform each activity. The matching between plan activities and *Actuators* is made using the resource type taxonomy previously mentioned. *Actuators* interpret the generic activities defined by the model into commands of the languages supported by the specific management interfaces. As an example, activity #5 (update a WAR artifact deployed at the Glassfish node3 cluster) is translated into the "DeploymentManager.redeploy" operation, which is defined



```
<?xml version="1.0" encoding="ASCII"?>
<plan:plan
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:plan="http://model.deployment.itecban.es/DeploymentPlan">
    <name>Test Plan</name>
    <environment>Target</environment>
    <activities>
<activity xsi:type="plan:DeploymentActivityType">
    <activityId>1</activityId>
        <target>target#node1#BPEL-PM</target>
        <type>INSTALL_DEPLOYMENT_UNIT</type>
        <unitName>ApprovalProcess</unitName>
        <unitVersion>2.1</unitVersion>
</activity>
<activity xsi:type="plan:ConfigurationActivityType">
    <activityId>2</activityId>
        <target>target#node1#BPEL-PM#ApprovalProcess#endPoint</target>
        <dependency>1</dependency>
        <type>CONFIGURE_UNIT_RESOURCE</type>
        <configuration>
            <property>
                <name>endPointURL</name>
                <value>http://machine1:8080/BPMEndpoint/</value>
            </property>
        </configuration>
</activity>
...
<activity xsi:type="plan:DeploymentActivityType">
    <activityId>6</activityId>
        <target>target#node2#OracleDB</target>
        <type>INSTALL_DEPLOYMENT_UNIT</type>
        <unitName>ProductCatalog</unitName>
        <unitVersion>1.6</unitVersion>
</activity>
...
<activity xsi:type="plan:DeploymentActivityType">
    <activityId>5</activityId>
        <target>target#node3#glassFish</target>
        <dependency>2</dependency>
        <dependency>6</dependency>
        <type>UPDATE_DEPLOYMENT_UNIT</type>
        <unitName>FrontEnd</unitName>
        <unitVersion>1.6.1</unitVersion>
</activity>
    </activities>
    <scheduleData>
        <creationDate>2010-01-22T10:59:01.250+01:00</creationDate>
        <lastModificationDate>2010-01-22T10:59:02.968+01:00</lastModificationDate>
    </scheduleData>
    <planStatus>CREATED</planStatus>
</plan:plan>
```

Figure 3 Generated Deployment and configuration plan Model

by the JSR 88 Deployment Management API offered by the Glassfish server. Finally, the *Change Executor* invokes the *Actuators* in the order dictated by plan dependencies, and as a result the runtime environment configuration is updated to support the new version of the credit grant service. Additional details on the instrumentation architecture are presented in [12].

VI. CONCLUSIONS

In this paper we have introduced the usage of dynamically generated models for enabling autonomic management of service oriented architectures. The proposed abstractions define both static models for descriptions of software components before they are deployed to the runtime environment, and dynamically generated models (obtained by monitoring agents) for the runtime environment elements. By mixing, transforming, aggregating and processing these models, we obtain a third model, the deployment and configuration plan, which is generated and consumed at runtime. Controlling agents are in charge of interpreting and executing it so the environment is changed accordingly to the plan.

Our interpretation of MDM offers several improvements to the state of the art: there are clear, unambiguous but extensible metamodels for all the information handled; static relations between models are represented at the metamodel stage, while relations between dynamic models are ensured by the proper transformations; transformations can take several models to produce a new one (as expressed in); agents are considered as producers or consumers of models, thus behaving as the frontiers of the management system; the runtime system can be observed at a high level of abstraction, at its logical view, but at the same time its model represents a real situation, the system “as it is”.

Once the models have reached a certain degree of maturity, and we have implemented a proof of concept of the system, we consider the whole set composed by metamodels, transformations, implementation of management functions and monitoring and control agents, as a complete infrastructure for the management of service oriented architectures. After successive versions of the system we have provided a method for the development of management functions, based on the provision of transformation on models, and the generation, adaptation or usage of the required agents for monitoring and control.

We are currently adding more management functions that will cover the full range of activities related to services lifecycle. In its industrial application it is also necessary to provide enhancements such as security and access control, persistence and management of the intermediate and final models, instrumentation for virtualized systems, generation of reports for business intelligence and integration with IT service level frameworks.

ACKNOWLEDGMENT

The work presented in this article has been partially funded by the ITEA OSAMI-Commons and CENIT ITECBAN R&D projects. The authors are grateful to MITYC (Ministerio de Industria, Turismo y Comercio) and CDTI (Centro para el Desarrollo Tecnológico e Industrial) for its support.

REFERENCES

[1] D. Schmidt, “Model-Driven Engineering”, IEEE Computer, vol. 39, no. 2, February, 2006.

[2] L.A. Fernandes, B.H. Neto, V. Fagundes, G. Zimbrao, J.M. de Souza, and R. Salvador, “Model-Driven Architecture Approach for Data Warehouse”, Proceedings of the 6th International Conference on Autonomic and Autonomous Systems, pp. 156-161, Cancun, Mexico, 2010.

[3] M. Barbero, F. Jouault, and J. Bézivin, “Model Driven Management of Complex Systems: Implementing the Macroscope's Vision”, Proceedings of the 15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems, IEEE Computer Society, 2008.

[4] V. Talwar, D. Milojicic, W. Qinyi, C. Pu, W. Yan, and G. Jung. “Approaches for service deployment” IEEE Internet Computing Magazine, vol. 9 Issue 2, pp.70-80, 2005.

[5] Hegering, H., Abeck, S., Neumair, B, “Integrated Management of Networked Systems. Concepts, Architectures and Their Operational Applications”, Morgan Kaufmann publishers, ISBN: 3-932588-16-9, 1999

[6] J.M. Gonzalez, J.A. Lozano, and A. Castro, “Autonomic System Administration. A Testbed on Autonomics”, Proceedings of the 5th International Conference on Autonomic and Autonomous Systems, pp. 117-122, Valencia, Spain, 2009.

[7] K. Wilson and I. Sedukhin, “Web Services Distributed Management: Management Of Web Services (MOWS 1.1)”. OASIS, Standard 2006.

[8] DMTF (Distributed Management Task Force) Common Information Model (CIM) specification v2.19. . 2008

[9] Object Management Group. Deployment and Configuration of Distributed Component-based Applications Specification. Version 4.0. April 2006.

[10] J.L. Ruiz, J.C. Dueñas, and F. Cuadrado, "Model-based context-aware deployment of distributed systems” Communications Magazine, IEEE , vol.47, no.6, pp.164-171, June 2009

[11] J. Strassner, “Handbook of network and service administration” Ed. Elsevier, 2007, ISBN 978-0-444-52198-9.

[12] F. Cuadrado, R. Garcia-Carmona, A. Navas and J.C. Dueñas, ”A Change Execution System for Enterprise Services with Compensation Support”, Conference on Enterprise Information Systems,. Madeira, Portugal. Communications and Computer Series. Vol. 109 pp.441-450. 2010

.