A model for enabling context-adapted deployment and configuration operations for the banking environment

Félix Cuadrado*, Juan C. Dueñas*, Rodrigo García*, José L. Ruiz**

*Universidad Politécnica de Madrid, ETSI Telecomunicación,
Ciudad Universitaria s/n. 28040, Madrid, Spain
** Indra, c/José Echegaray 8,28108 - Parque Empresarial - Las Rozas, Madrid, Spain
*{jcduenas, fcuadrado, rodrigo}@dit.upm.es,**jlrrevuelta@indra.es

Abstract*-In the banking sector business requirements continuously change whereas IT infrastructure investments must be amortized over years. This conflict produces very heterogeneous systems. Adopting the SOA / BPM approach helps coping with that complexity. This way, everything is a service, easing composition and integration. On top of that, strict security and reliability requirements exacerbate the need of a robust management infrastructure. We propose a service-centric operational management architecture for which we have defined a resource model based on the leading standards for characterizing systems, services and operations. This model is supported by a dynamic agent infrastructure, which automatically instruments the targeted environments. These concepts are illustrated by a proof of concept consisting of deployment and configuration activities over distributed banking services.*

Keywords*- Service-oriented architectures, Service instrumentation, Service Management, Context adaptation*

1. Introduction

In the banking sector investments in legacy systems must be amortized over long periods of time. On top of that, it is necessary to upgrade applications and services, and adopt new technologies for B2B and presentation services. Thus, systems are composed by not only legacy systems, mainframes, databases, but also JEE application servers, or BRM (Business Rule Managers) systems. The SOA / BPM approach [1] is the preferred way of operating these heterogeneous systems. 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 ESB (Enterprise Service Bus). On top of that, Business Process Management technologies, such as BPEL (Business Process Execution Language), orchestrate the activities, bridging the gap between the IT infrastructure and the business processes.

Competitiveness and innovation demand releasing frequent upgrades and new services. At the same time, developed artifacts must comply with high standards of security and reliability. These requirements fall under the management architecture, which must support monitoring accurately the system, reacting to unexpected behavior, as well as deploying new services, upgrading system artifacts or executing configuration activities. All in all, it must deal with all the heterogeneity and complexity of the banking infrastructure, as well as the abstractions provided by the service layer used for operation.

These problems are not new, and have been addressed by several tools and standards. The main problem with available commercial service management suites is that, while they successfully manage those environments, they frequently force users to adopt their own product family for the base system infrastructure. On the other hand, network and system management tools are not well suited to managing these runtime services, focusing on the base infrastructure instead.

Because of that, we have developed a service-centric management architecture for distributed services. This paper describes our information model for characterizing manageable services, and a monitoring infrastructure for adaptation to the context environment. First, we present a brief overview on the relevant standards and initiatives in the field of software and services management. Our contributions are further described through a proof of concept, consisting of deployment and configuration of a distributed application over a reference banking environment.

2. State of the art

2.1 Service Management Information Models

There are several information models targeted at the description of heterogeneous distributed environments. The CIM (Common Information Model) [2] is an object-oriented model for describing overall management information in a networked enterprise environment, maintained by the DMTF (Distributed Management Task Force). CIM is structured as a core model, defining the basic elements, and extensions for detailing parts of the system, such as databases, networks, applications, software products and devices. The depth of the modeling and the granularity of the standard usually mean CIM-compliant tools support selected profiles or custom extensions from the base model.

The OMG Deployment and Configuration (D&C) specification [3] provides a model for representing deployment and configuration operations over a distributed target. The model is object-oriented, simple and flexible. D&C base elements are resources, which are named entities classified into one or more types. Resource instances model physical artifacts, such as nodes, bridges and links. Resources are parameterized with a collection of properties. Each property has a name, a value and a kind, which determines the consumption nature of the resource. The combination of types and properties enables resource managers to operate on heterogeneous environments working with the same base concepts. However, the model is focused on modeling network and hardware resources instead of service management elements.

MUWS (Management Using Web Services) [4] is a standard of the OASIS WSDM group, aiming at describing and managing resources through Web Services. In MUWS each manageable element of a distributed system is modeled as a resource. Manageable resources have a well-defined set of operations, known as capabilities. The specification defines basic capabilities, such as description (which allows to obtain resource's name and version), state, metrics or configuration (through properties configuration). This mechanism is extensible, allowing some resources to expose specific management interfaces in addition to the basic ones.

2.2 Heterogeneous system management

Traditional management processes involve human operations over a management administration console. However, the increase of complexity, distribution and heterogeneity of current IT systems is stressing the limits of this approach. On top of the aforementioned domain models, management tools must aggregate the runtime information of the system and apply the necessary operations automatically. That is the reason for the surge of the autonomic computing paradigm [6]. A self-managing system can greatly reduce its operation cost and perform automatically well-known management processes. Autonomic managers implement an intelligent control loop for automating at least some of the management aspects of a resource. For environment-wide operations managers can be orchestrated, and still be manually managed in some critical cases.

The PBM (Policy-Based Management) [5] presents a complementary approach for automating system management based on the use of policies. Policies are usually defined as rules, which allow reasoning over the collected information and invoke operations on the management interfaces. This combination provides a simple mechanism for modifying the behavior of the autonomic manager. PMAC (Policy Management for Autonomic Computing) [7] is an example platform which leverages policies, written in their own language, ACEL to an autonomic manager. A similar approach is adopted by Focale [8], an autonomic manager implemented with ontology-based policies.

In our scenario we would need to define configuration and deployment policies for a distributed system where multiple services from heterogeneous sources are composed to provide functionality. But before that, we need to define a common information model or ontology. So far, none of the evaluated standards provide a unified framework for achieving that.

3. Management Model and Architecture

Our proposed deployment and configuration architecture is envisioned to work over heterogeneous environments. It also must be able to dynamically adapt to changes in those environments without manual intervention. These high-level requirements have been detailed and expanded into a set of use cases for the complete management architecture, such as:

- Increase the size of a cluster of servers (and automatically replicate the deployed services on the new node) to maintain a SLA (Service-Level Agreement) over an increased number of requests.
- Deploy a new service to the environment. The service is provided by several elements deployed to different nodes. The operation is executed in a transactional manner, with a feedback channel guaranteeing the stability of

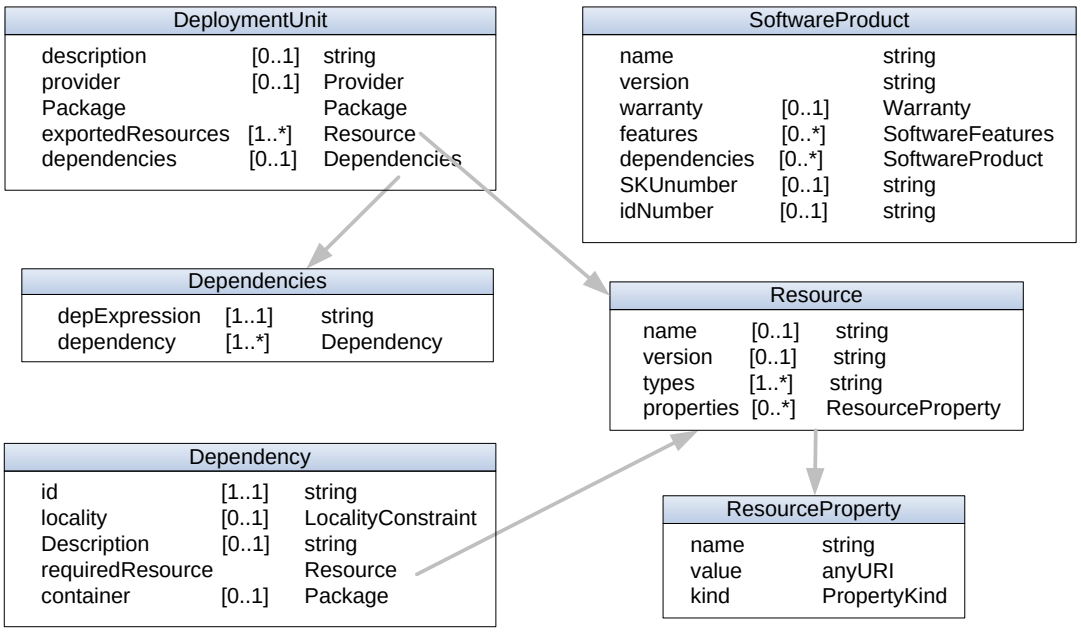

Figure 1 Resource and software model

the system during the operation, and reverting back to initial state on the occurrence of failures.

- On the appearance of an additional implementation of a service, configure a load balancer to redistribute service requests

In order to fulfill these scenarios the context adaptation layer plays a fundamental role. This section describes its two main elements: a common resource model, shared by every entity, suitable for heterogeneous systems and services, and an instrumentation infrastructure bridging the gap from the model to the specifics of each vendor technology.

3.1 Resource model

After the analysis of some of the most important resource information models and management approaches, we have identified common ground in the concept of manageable resources, both in D&C and MUWS standards, as the root of their models. However, for managing heterogeneous applications and services the standards do not provide a complete solution. For example, the lack of resource versioning information limits the capability to handle services and their dependencies correctly. On top of that, we need a model able to integrate resources both from the environment and the services implementations. We have defined our model based on those standards and implementing these additional requirements, part of which can be seen in Figure 1.

The main elements of the model, defined in XML Schema, are resources, defined as manageable entities of the system.. A resource has a name, a version identifier and a set of properties for its complete characterization. In addition to that, resources are classified into types, which allows management systems to define actuators and policies which automatically apply to the matching elements. The model definition is complemented by a resource taxonomy, for characterizing the basic assets of a banking production environment (ranging from services to containers).

On top of this base element we have modeled both the operation environments, and the applications and services. Both models are linked through resources.

The software model focuses on describing deployable software and services. As services are runtime entities they are clearly modeled as resources in our information model. The model provides a detailed description of the software artifact that provides the runtime service (in our terminology, a deployment unit). A deployment unit definition includes its logical dependencies, as well as constraints on the environment in order for the unit to be installed and work correctly (e.g. minimum amount of RAM memory). The model adopts the main elements of CIM Application Model for describing software artifacts, ranging from Software Products to the low-level resources available at runtime.

On the other hand, the environment model describes the topology of the environment, using the D&C target data model. Environments are composed by nodes, interconnects and bridges. These elements are further characterized by resources, extended in our model with version information. In addition to that, we have added containers hosted by nodes, an additional element for representing the containers of applications and services, as they play a vital role in services configuration and deployment.

Then, services are described by means of the software model which contains both resource requirements (needs of the services implementations), and resource offerings (parts of services implementations that can be used by other ones). The operation environment is also described by the model –that can ultimately be transformed into a set of offered resources-. A model-based management architecture needs to provide two additional functions: the capability to get information about the operation environment in execution (we call this agent infrastructure); and the component able to match the resources offered by this environment with the resource requirements given by each piece of services implementation.

3.2 The agent infrastructure

The agent infrastructure must instrument dynamic heterogeneous systems, by means of translating the context-specific information to our generic model. On top of that, it must react to changes to the topology of the environment. So, agents must have a mechanism for automatically aggregating information from new elements. However, manual configuration mechanisms must also be supported for agents providing critical information, which can't be automatically integrated, due to environment firewalls.

With these requirements in mind, we have designed a layered infrastructure for instrumenting the target environment. Its main features are automatic aggregation of agents and the capability to automatically adapt to context changes. It is composed by several elements:

The Node Manager: manages the capabilities/resources available at the node it is handling. Hardware and software resources are read, monitored and managed thanks to the association with the *ContextGatherer*. In addition, it executes deployment and configuration operations: installation, activation, deactivation, (re)configuration, removal and update of deployment units.

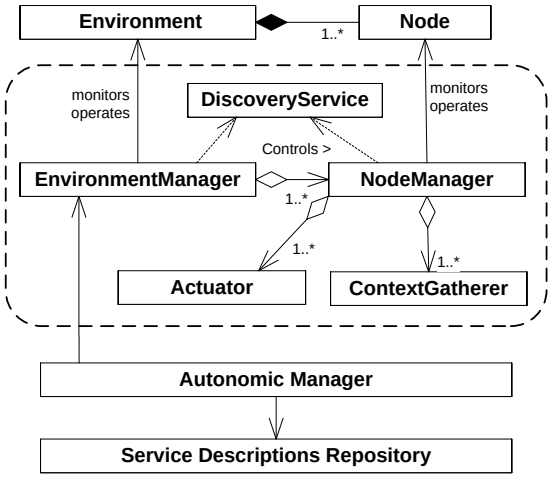

Figure 2 Instrumentation Infrastructure

The Context Gatherer: collects node resource information and exposes this information through well-defined services using the resource model. Information can be provided by sources of different nature, e.g. operating system details (version, name, libraries installed and so on), hardware resources (static capacities and available free resources), container configuration, battery life, etc. There can be a different context gatherer for each type of context (hardware, operating system, container, services implementations, etc). The adaptor design pattern has been used to create a progressive and scalable implementation of the Context Gatherer. In order to avoid excessive resource consumption during this process an event driven model for the communication among context agents and the gatherer is applied.

The Actuator: performs deployment and configuration operations on specific resources / containers of the environment. Each actuator executes one or more operations (i.e. configure container, install deployment units) on some parts of the environment .The taxonomy allows matching containers to compatible actuators.
The Environment Manager: coordinates the activities that take place at the environment. It communicates with the existing node managers to provide an aggregated monitoring view of the system and delegate the deployment and configuration activities to the specific actuators. It is aware of node manager instances running on the deployment target, as well as its capabilities. This information can also be manually configured by an administrator, and stored into a configuration database. However, creating the description of the environment is usually an arduous task. And even worse, descriptions have to be updated according to changes in the target. We have automated this process by means of including a discovery

mechanism based on DNS-SD (Service Discovery), so it detects node managers as they come and go and therefore avoid most manual configuration operations. The manager also ensures each discovered node manager is working properly, sending a fault message and removing it from its list otherwise.

4. Case Study

In the ITECBAN project we needed a configuration and deployment architecture matching the demands of heterogeneous banking systems. As a reference scenario, we work over a SOA / BPM banking services platform composed of several interconnected services. The target environment is composed of three nodes with several execution containers provisioned, both commercial and open source: JEE application servers (BEA Weblogic and JBoss), an ESB (Apache ServiceMix), a database (Oracle 10g), a BPM engine (Oracle BPEL Process Manager) and a BRM repository (Drools BRMS). Over this scenario a reference banking service has been developed. The banking service is composed by five components (war files, rule packages, DDL data definition files and BPEL business processes), which must be deployed over containers, as shown in the figure 3.

In this case study we use the instrumentation infrastructure to enable distributed deployment of the banking services implementations. The operation is executed transactionally, ensuring the stability of the managed environment. The environment manager and node managers provide the topology information. Context gatherers contribute information about the installed containers and the available resources. Actuators are registered for the deployment operations on each different containers. We can see a short fragment of the automatically generated environment snapshot in the following listing.

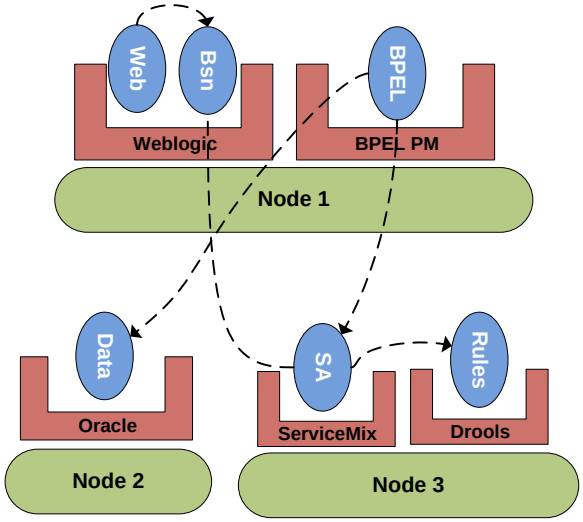

Figure 3 Logical view of the banking service

```
<node>
   <name>node1/ip</name>
   <nodeContainers> <nodeContainer>
      <name>node1-weblogic</name>
      <containerTypes><containerType>
           <name>es.itecban.deployment.
     container.jee</name>
           <version>5.0</version>
      </containerType></containerTypes>
      <supportedPackages>
            <containerPackage>
           <type>es.itecban.deployment.
packaging.war</type>
            </containerPackage> …
      </supportedPackages>
      <containerResources>
            ...

</containerResources></nodeContainers>
</node>
```

Additionally, each individual software component has been described based on the resource model, expressing service dependencies and environment constraints. The deployment manager processes these models, and performs a resource matching between the software dependencies and the environment. After the models have been processed, an installation plan adapted to the specific environment is obtained. The plan is composed by a set of activities which will be performed by each installer, involving several containers of the environment. The use of context information in the plan creation allows optimizing the list of activities, skipping the installation of components already present at the environment.

The deployment manager uses both sensor and actuator channels of the environment manager, closing an intelligent loop which allows deploying services implementations. Each operation includes a verification check for validating if it has been correctly executed, based on the observed changes to the environment. If at any stage of the plan a problem appears, the manager undoes every registered step in order to go back to the initial stable state.

The example illustrates the main advantages of our approach. By using the same base model for the description of both the services implementations and the target environment we can automate the management operations, which follow this basic cycle:

1. Receive requests for changes to the environment, referred to services elements, and described as sets of resources requirements, and sets of new resources offered by the services elements
2. Collect information about the environment, identify the set of resources offered by the environment and present it using the resources model
3. Match resources required by the services elements with the resources offered by the environment
4. Search for new services elements that fulfill resources requirements not completed by the environment, until all the resources requirements are filled

Thus, we can say that the infrastructure adapts automatically to the environment, and the model provides the common base for the deployment and configuration manager. This way, the manager can automatically create a change plan tailored to the service and the runtime state of the environment. The manager esolves service dependencies, maps installable artifacts to valid nodes from the environment, and creates the configuration operations for a correct service binding. The model plan is carried out by the designed actuators, translating it to specific operations from the systems. Most of these operations can be performed automatically, except in the case of not being able to complete the resources requirements, where human intervention would be required.

5. Conclusions

We have created a resource-centric model for describing services as well as its environment. This common ground for both services and systems supports an unified view, and transversal logic among the two worlds. In addition to that, the resource taxonomy enables automatically matching dependencies, as well as actuator agents to the containers of the environment. Besides, the agent infrastructure mediates between the model and the specific details of the environment. Automatic discovery and aggregation mechanisms greatly improve their reusability on different environments (or dynamics) without additional configuration.

Our case study shows how our instrumentation architecture enables a deployment and configuration distributed control system, taking to practice autonomic principles during the execution of a high-level system-wide activity (deploying a banking service).

Future work will be oriented at extending the deployment and configuration manager in order to improve management of the whole services lifecycle, by extending the functionality of actuators. Also, the model can be extended to support the management of virtual environments. This way, it would be possible to operate at the topology level for coping with changing requirements (for instance, creating a new virtual node, provisioned with another server, where the service is installed, and configuring a load balancer to distribute the traffic).

Acknowledgements

ITECBAN is an IT innovation project partially funded by CENIT (a Spanish public R&D program). The authors are grateful to MITYC (Ministerio de Industria, Turismo y Comercio) and CDTI (Centro para el Desarrollo Tecnológico e Industrial) for supporting ITECBAN through CENIT and Indra as the main contractor of the project.

References

[1] S. Brahe, BPM on Top of SOA: Experiences from the Financial Industry, Lecture Notes on Computer Science 4714, pp. 96-111, ISSN 0302-9743 2007.

[2] Information Model (CIM) specification v2.1, DMTF standard, http://www.dmtf.org/standards/cim

[3] Deployment and Configuration of Component-based Distributed Applications Specification. OMG formal specification version 4.0 formal/06-04-02, 2006.

[4] V. Bullard, W.Vambenepe, WSDM, Web services distributed management: Management using Web services (MUWS 1.1), OASIS Standard, August 2006. Available at http://www.oasis-open.org/committees/download.php/20576/wsdm-muws1-1.1-spec-os-01.pdf

[5] D. C. Verma, Simplifying Network Administration Using Policy-Based Management, IEEE Network, March 2002

[6] An architectural blueprint for autonomic computing, IBM Whitepaper, available at http://www-03.ibm.com/autonomic/pdfs/AC_Blueprint_White_Paper_4th.pdf

[7] B. Jennings, S. van der Meer, S. Balasubramaniam, D. Botvich, M. O. Foghlú, W- Donnelly and J. Strassner, “Towards Autonomic Management of Communications Networks”, IEEE Communications, Oct 2007

[8] D. Agrawal, K. Lee and J.Lobo, “Policy-Based Management of Networked Computing Systems”, IEEE Communications Oct 2005.