A Change Execution System for Enterprise Services with Compensation Support

Félix Cuadrado[1], Rodrigo García-Carmona[1], Álvaro Navas[1], Juan C. Dueñas[1]

[1] Universidad Politécnica de Madrid, ETSI Telecomunicación,
Ciudad Universitaria s/n. 28040, Madrid, Spain
{fcuadrado, jcduenas, rodrigo, anavas}@dit.upm.es

Abstract. Modern enterprises rely on a distributed IT infrastructure to execute their business processes, adopting Service Oriented Architectures in order to improve the flexibility and ease of adaptation of their functions. Nowadays this is a vital characteristic, as the increased competition forces companies to continuously evolve and adapt. SOA applications must be supported by management and deployment systems, which have to continuously apply modifications to the distributed infrastructure. This article presents a model-based solution for automatically applying change plans to heterogeneous enterprise managed environments. The proposed solution uses models which describe in an abstract language the changes that need to be applied to the environment, and executes all the required operations to the specific managed elements. Also, to ensure that the environment ends in a stable state, compensation for previously executed operations is supported. The validation results from a case study taken from the banking domain are also presented here.

Keywords: Service deployment, Plan execution, Enterprise Infrastructure, Compensation Support, Model-Based Management.

1 Introduction

In a globalized world enterprises have to face greatly increased competition, demanding agility to release new products and update to customer demands. These factors have lead to the adoption of the service oriented paradigm. This paradigm produces execution infrastructures composed by multiple, heterogeneous servers with specialized roles, distributed over a network. This setup greatly complicates technical management processes, such as diagnosing the environment status, planning the required changes or applying corrections to improve its performance.

Frequently those tasks are executed manually by an IT administrator, but this approach is very costly and hampers the desired agility. Therefore, an increased degree of automation in service change management operations is a must for obtaining the potential advantages of the service oriented approach. A change execution system that coordinates operations over the distributed servers and containers is needed. However, this is complicated by the fact that the actual

composition of the managed environment is unknown at design time (both in number of elements and types of infrastructure systems. This can be addressed by adopting a model-based abstraction layer for describing the operations and the runtime elements. This way, the resulting system can flexibly adapt to the specifics of each environment.

At the same time, the critical non-functional requirements for enterprise systems need also to be supported by these management functions: The stability of the managed environment must be preserved by the operation systems, as otherwise the economic impact would be enormous. This way, if during a change execution operation an unexpected event occurs, the incidence should be detected and the system should try to restore the original state.

This paper presents a model-based architecture of a system for applying deployment and configuration changes to a complex enterprise system. The presented solution automatically adapts to the characteristics of each targeted environment, combining an extensible architecture with the use of model abstractions for representing the operations. On top of that, the actual execution of the changes is carried out with a fine control thanks to business process technologies. This enables the definition of an algorithm for inverse operation identification and compensation plan dynamic creation. The system has been developed with the concerns of the enterprise banking domain under the context of the ITECBAN project. This project combines the research efforts of banking and consulting companies, as well as academia, to try to provide a complete core banking service oriented solution.

The structure of the paper is as follows. Next section provides an overview on the most relevant initiatives related to deployment and configuration changes execution over distributed environments. Section three describes the model needed to support the system. The next section details the most relevant aspects of the decided architecture. Once the proposed solution has been completely described, section 5 presents a case study used to validate these contributions. Finally, the paper finishes with some general conclusions possible continuation for this work.

2 Related Approaches on Distributed Configuration Activities Execution

Automated execution of changes into a distributed heterogeneous environment represents a research challenge which has been addressed from different approaches. Although most of them follow the Information Technologies Infrastructure Library (ITIL) definition of the change management process [1], they differ on how to implement that definition. One of the more important aspects is how to define the change plan, containing the operations that will lead from a stable state to the state specified by the Request for Change (RFC) submission.

There are other problems related to the definition of a plan and how to implement it. First, an abstract definition of the operations is needed so the model is valid in heterogeneous environments. Another critical problem related with defining a plan is how to determine the order in which each operation must be carried out.

Regarding the plan definition, the Object Management Group (OMG) provides, in Deployment & Configuration of Component-based Distributed Applications, v4.0 [2],

a model for the deployment plan, which is implemented by some systems such as Darca[3]. However, this specification is too centered on installation activities in homogeneous environments, lacking support for both the management of the whole artifact life cycle and establishing an order in which the activities must be performed.

Implementations of change execution managers offer a wide range of solutions. Champs [4] proposes the use of a temporal planner to create plans, taking in account the defined policies and Service Level Agreements (SLA). Plans are defined using Business Process Execution Language for WebServices (WS-BPEL) [5]. The change plan tries to maximize parallel execution of activities, thanks to the use of Constraint Satisfaction Problem techniques. This is done at the cost of flexibility, not being able to adapt to errors or changes during execution.

Other systems, such as ChangeLedge [6], propose a model which considers that operations can go wrong, and systems can make a rollback. In ChangeLedge, when a plan fails, the system automatically stops the execution and creates a rollback plan defined in BPEL[7], forcing each atomic action to rollback. On top of that, ChangeLedge adds the remediation plan concept [8], where the Change Plan designer has the opportunity to define an alternative plan that is executed automatically when one of the reversible activities fails to complete, along with the previous support for rollback. The main problem with Changeledge is the increased human effort in the design phase, as a human operator must not only to specify the RFC submission, but also to complete the definition of the change plan, and to design the rollback and remediation actions.

Other approaches use a planner based on absolute time. The PlanIT system [9] is an automatic configuration change planner for distributed systems that uses the Planning Domain Definition Language (PDDL) for defining components, environments and plans. Each activity in the change plan is defined both by the description of what it does and the absolute time at which it must be executed.

Ecotopia [10] is a framework that tries to minimize the service-delivery disruptions caused by changes, producing a change plan in which the activities are executed when they cause less impact on the system. This absolute temporal planning approach requires knowing with a high degree of confidence the estimated time to complete each activity, but the uncertainty which characterizes enterprise environments makes this approach unfeasible.

Finally, the most common approach consists in using manually configured scripts, such as Apache Ant, to define and execute the tasks that must be performed.

3 A Model-based Deployment and Configuration Architecture

Our research inside the ITECBAN project has been focused on improving the operation processes for the complete lifecycle of banking services, such as provisioning of updated components, replacement of no longer needed ones, or decommission after its complete working period has expired. To support these functions a deployment and configuration architecture has been designed and implemented. This architecture needed to reason about the management environment generically, without being tied to a specific platform or service type. To achieve it, we

used models as an abstraction layer to the real elements of the system. This way, the operations are performed by a set of loosely coupled services that communicate through model instances.

A typical example of these functions is the deployment of a new service to the managed environment. This is achieved by the architecture through the invocation of several components which, starting from an initial objective (the provisioning of the service), perform tasks such as connecting to the instrumentation agents for retrieving the runtime information, accessing software repositories, deciding which compatible version of a service to use or choosing where each deployable artifact should be physically located. As a result, a *Deployment Plan* with these tasks is produced. This plan is a model specifying what changes must be applied to the environment in order to achieve the desired objective.

The change execution service takes a plan as an input and applies the changes defined in it to the managed environment. This service must support the heterogeneity of the execution platform, since the plan is composed of multiple operations over the distributed hardware and middleware elements whose exact nature is not known beforehand. Therefore the change execution service must be flexible and extensible enough to adapt to the environment composition.

Also, the operations executed by this service must leave the environment in a stable state.

3.1 The Deployment Plan Model

The *Deployment Plan* model allows to define what changes must be applied to the managed environment in order to achieve a business objective: Each operation, the physical elements which are affected, and the constraints for applying them correctly. Its metamodel is defined in EMF (Eclipse Modeling Framework) Ecore [11], an implementation of EMOF (Essential MOF). The next picture shows the main elements of the metamodel.

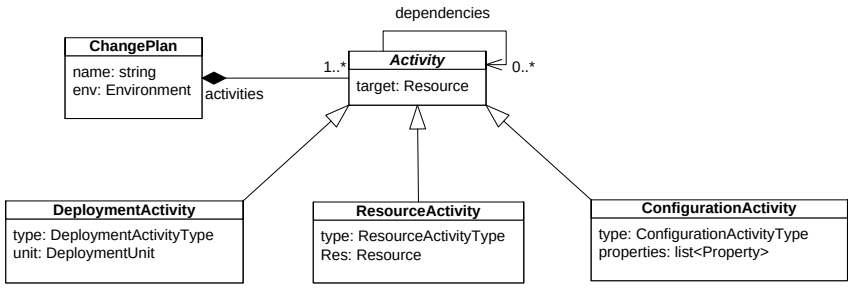

Fig. 1 Deployment Plan metamodel

The root entity is the *Deployment Plan*, which is uniquely identified by a name, and explicitly refers to the environment where it will be executed. A plan is composed by a set of *Activities*, which correspond to the change operations that can be initiated by the management architecture. Each *Activity* identifies the target *Resource* from the

runtime environment where it will be applied (application servers, EAR files, Web Services, etc). There are ten specific change operations which can be included as part of a plan, which have been grouped into three *Activity* subclasses. Each subclass allows the identification of the specific type of primitive (e.g. installation instead of update or uninstallation), provides additional information about the required execution parameters and restricts which resources can be targeted.

Deployment Activities control the life cycle of the runtime deployed artifacts. They include installation, activation, update, deactivation, and uninstallation. *Resource Activities* allow the creation or removal of resources of runtime containers, like application servers or databases. Finally, *Configuration Activities* modify the configuration of existing resources updating their properties.

In addition to defining the *Activities,* it is necessary to provide a mechanism to restrict how they must be executed. In the state of the art analysis multiple mechanisms were presented for linking the plan activities. Considering the heterogeneous nature of the elements, the simplest mechanism has been selected: Each *Activity* can identify any number of *Activities* as dependant ones, meaning that its execution will not start before all of them have finished theirs. This is identical to the Ant target dependency. Provides enough expressivity to know if the execution has been correct and is generic enough to be applied to a great variety of activities.

4 Change Plan Executor Architecture

The Change Plan Executor has been designed as a service-based application. This is supported not only by the adoption of Web Services as the remote communication mechanism but also by using the OSGi platform [12. for its internal structure. This framework provides a powerful component model and a local service registry where dynamic registration and binding of services can be achieved.

The system is composed by three collaborating elements which implement the complete plan execution process, as is shown in Fig. 2 The *plan parser* service sorts the plan activities and builds an execution flow which respects their dependencies. Each node from the flow is associated to a different *executor* service which can apply the specific changes to the targeted system. The *execution service* controls the application of the changes defined in the flow, verifying their correction and respecting the set order.

For increasing control over its application, the execution flow has been modeled internally as a process using the Process Virtual Machine (PVM) language [13. . This language serves as a metamodel in JBPM (JBoss Business Process Management) for defining specific process languages such as jPDL or BPEL. It contains only the base process concepts (nodes, transitions) and provides a process execution service which allows rich control over the process execution. This way, the benefits of business process approaches are obtained without having to adopt an excessively complex language (such as BPEL) which would needlessly complicate the internal model. Therefore, the change process is composed by nodes and transitions, with each node representing the execution of a change plan activity over a runtime target (e.g. deploy

a WAR artifact to a Glassfish application server, or configure the service port of an Apache http server).

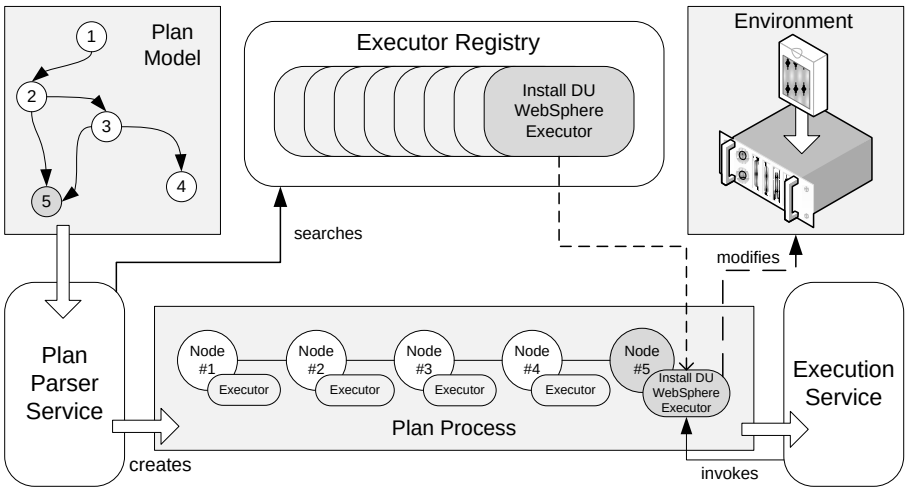

Fig. 2 Change Execution System Architecture

In order to generate a PVM process the *plan parser* service takes as input the deployment plan (a graph with multiple ways to process it) and produces a sequence with only one way of traversing it. This way the execution service knows in which exact order has to process it and what activities have already been executed. This is especially important for the compensation mechanism described in the next section.

Each process node is related to one *executor* service. These services are entities capable of executing a plan activity type in a specific container. They translate the generic activities into specific operations (e.g. invoking an Ant script, or connecting to the management interface of a server). These elements are published in the internal registry. The *parser* automatically matches plan activities with the available *executors* and associates a valid *executor* to each process node. This approach enables the *executor* architecture to be extensible and automatically adapt to plans executed over different environments with heterogeneous technologies.

Finally, the *execution service* controls the application of the defined change processes, by orchestrating the invocation of the *executors* associated to each node. Along this process, it also generates a report on the result of the plan execution, aggregating the outcome of each activity (generated by each *executor*). The service is highly flexible in the execution mechanism, supporting multiple execution modes including human-controlled operation (with an administrator invoking each step of the process) and completely automated execution with and without compensation execution. Because of its relevance, we will detail the internal mechanism for supporting automatic plan compensation execution.

In order for the change execution service to respect the system stability, it is mandatory to ensure that the applied changes do not negatively affect the state of the environment. This has been supported by providing compensation execution

capabilities to the system. This characteristic of the architecture is supported by two mandatory capabilities of every *executor*: 1) Their execution is atomic, 2) The result of their execution is notified (successful or not, informing in the latter case about the type of error). These requirements enable us to know exactly in which point of its execution a plan has failed and in which state it is.

The compensation module starts to work whenever any of the process *executors* reports an error on its execution. Since the execution has been sorted as a sequence of operations, in order to compensate its results, it is necessary to reverse the changes from the activities which have been already carried out. The activity that has failed does not need to be compensated since, being atomic, no change has been performed.

The task of determining what operation will counter each applied change is nontrivial, but it can be automated thanks to the defined plan model. Each one of the ten primitives which can appear at a deployment plan has been formally defined, including the required arguments, and the effect it will cause on the managed environment (e.g. deploying a new artifact over a container will cause a new web application to exist). By looking at that information, as well as the initial state of the environment (which is also defined through an information model), it is possible to automatically obtain an opposite activity for each one defined at the plan, The following table presents an overview of the complete set of supported operations and their compensation activities:

Table 1. Change Plan Compensation Activities

Original Operation	Arguments	Compensation
Install Unit	Unit, container	Uninstall Unit
Update Unit	Unit, container	Update Unit
Uninstall Unit	Unit, container	Install Unit
Start Unit	Unit, container	Stop Unit
Stop Unit	Unit, container	Start Unit
Add Resource	Resource, properties, container	Remove Resource
Remove Resource	Resource, container	Add Resource
Config Resource	Resource, properties, container	Config Resource
Config Unit Properties	Unit, properties, container	Config Unit Properties
Config Unit Bindings	Unit, properties, container	Config Unit Bindings

Plan compensation is supported at runtime by dynamically modifying the PVM process after detecting an execution fault in an activity. This way, after the execution finishes the environment will be restored to its initial state. In order to do so, the compensation module first removes the pending nodes. After that, for each successfully executed activity, an additional node is inserted in the process, following a reverse order sequence. Each node will be associated with an *executor* configured as the compensation operation for the one initially applied, both in its parameters and type. Once the process has been completely modified, the execution will proceed by invoking these new operations. Therefore, after the process is completely executed, the environment will be restored to the state it had prior to the execution.

This approach, however, has two limitations. First, an executor compatible with each compensation activity must be available in order for the compensation to work. In the event that another error occurs during the compensation execution the process

will stop, as it is not possible to automatically restore the system. A notification will be sent so that the IT personnel diagnose the unstable state.

5 Validation

After the system has been described in the previous sections, we will present the steps taken to validate our proposal. In order to do so, we will detail the results of a case study executed in a banking environment.

A banking company bases their business processes on a core banking system based on SOA principles. This system supports every company service, including B2B (business to business), bank staff services, end user internet banking and cashier operations. The specific services, such as credit concession or account management, are provided by components and services internally developed by the company personnel, under the guidelines of the internal SOA infrastructure. Services are deployed over the integration environment of the organization. The environment is composed by four computing nodes, which are provisioned with application servers, web servers, database systems, ESBs (Enterprise Service Buses) and BRMS (Business Rule Management Systems).

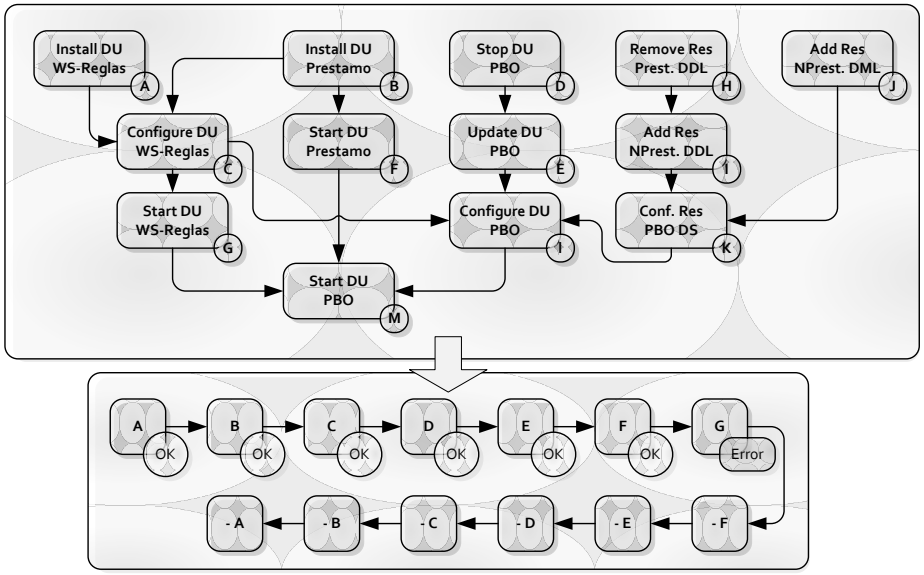

Fig. 3 Case Study Compensation Example

In the first execution everything proceeds smoothly. All the activities are executed correctly and an aggregated report is produced.

However, during the second execution the plan doesn't proceed correctly because the current state of the environment has changed from the moment where the plan was generated: we purposely shut down one node simulating a hardware malfunction.

Because of that, the first executor that had to apply changes to this node could not complete its task and therefore produced an error report. This happened during the start of the WS-Reglas deployment unit. When the plan launcher detected this, it invoked the compensation module to modify the plan process, inserting a compensation node for each one successfully applied beforehand. The resulting process is depicted in the previous figure. Compensation activities are represented with the minus sign (-) followed by the character of the original node it compensates.

Once the updated process was completely executed, we verified that the environment was, in fact, in the same state as before executing the plan. On top of that, a report explaining that a grave error had occurred at the environment was produced and passed to the administrator.

These two samples have shown that the change execution system perfomed as expected in a normal situation and in the case of an error during the launching of the change plan.

6 Conclusions

This paper has presented a complete solution for applying a set of related deployment and configuration operations onto heterogeneous distributed environments. The system seamlessly interacts between generic models and the specific managed system. The solution also provides compensation capabilities for any plan model provided. The abstraction at model level provides ten clear definitions for the potential atomic changes, which has allowed automatically defining and obtaining the compensation activity for each of them, enabling the automatic generation of the compensation plan. This is also supported by the internal design of the executor service, where the handling of executions as business processes provides a fine control over the whole process.

This approach however has several limitations which should be discussed. It has already been mentioned that in order for the compensation capabilities to be achieved with this technique, the operations from each specific interpreter must be applied atomically. This can prove to be a strong requirement on those agents, as it is heavily dependent on the specific characteristics of each runtime platform.

The proposed service is also not designed to optimize the total execution time of the provided deployment plans. As the model leaves some degree of flexibility to interpret it, it would be possible for instance to maximize the parallel execution of its activities, whenever possible. However, this increase in performance would impact the compensation capabilities, as it would be considerably more complex and error-prone that with the current, sequential approach. However, we intend to explore this line of evolution for our future work.

In addition to that, whenever the system tries applying changes over an environment which has been altered since the reasoning modules of the architecture obtained the required solutions, the only potential response of this component is to abort and restore the environment to the initial state. It would be interesting to see whether for those situations the reported error in the execution was handled to a

diagnosing module, which could potentially determine the source of the error, and modify on the fly the deployment plan in order for it to work correctly

Acknowledgments. The work presented here has been performed in the context of the CENIT-ITECBAN project, under a grant from CDTI (Centro para el Desarrollo Tecnológico Industrial) - MITYC (Ministerio de Industria, Comercio y Turismo de España), and the INDRA company as main contractor.

References

1. ITIL: Information Technology Infrastructure Library (ITIL). Office of Government Commerce (OGC), http://www.itil.co.uk/ (2006)
2. Deployment and Configuration of Component-based Distributed Applications, v4.0, http://www.omg.org
3. Dubus, J., Merle, P.: Applying OMG D&C Specification and ECA Rules for Autonomous Distributed Component-Based Systems. In: Models in Software Engineering, pp. 242-251. Springer Berlin/Heidelberg (2007)
4. Keller, A., Hellerstein, J.L., Wolf, JL, Wu, K-, Krishnan, V. The CHAMPS system: change management with planning and scheduling. Network Operations and Management Symposium (NOMS 2004). Vol.1. pp: 395-408 (2004).
5. BPEL4WS V1.1 specification, http://www.ibm.com/
6. Cordeiro , W. L. D. C., Machado, G. S., Andreis, F. G., dos Santos, A. D., Both, C. B., Gaspary, L. P., Granville, L. Z., Bartolini, C., Trastour, D.: CHANGELEDGE: Change design and planning in networked systems based on reuse of knowledge and automation. Computer Networks Volume 53, Issue 16, pp. 2782-2799 (2009).
7. Web Services Business Process Execution Language Version 2.0, http://www.oasis-open.org
8. Machado, G.S., Corediro, W. L. D. C., dos Santos, A.D., Wickboldt, J., Lunardi, R.C., Andreis, F.G., Both, C.B., Gaspary, L.P., Granville, L.Z., Trastour, D., Bartolini, C.: Refined failure remediation for IT change management systems. IFIP/IEEE International Symposium on Integrated Network Management (IM '09), pp 638 – 645 (2009).
9. Arshad, N., Heimbigner, D., Wolf, A.: Deployment and dynamic reconfiguration planning for distributed software systems. Software Quality Journal 2007, Vol. 15, Issue 3, pp:265-281 (2007).
10. Dumitraş, T., Roşu, D., Dan, A., Narasimhan P.: Ecotopia: An Ecological Framework for Change Management in Distributed Systems. Architecting Dependable Systems IV, Springer Berlin, pp 262-286 (2007).
11. Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: EMF: Eclipse Modelling Framework, Second Edition, Addison-Wesley Professional. ISBN 978-0321331885 (2008).
12. OSGi Service Platform Release 4 Core Specification, http://www.osgi.org
13. The Process Virtual Machine (PVM), http://docs.jboss.com/jbpm/pvm/article/