. L. Ruiz, J. C. Dueñas and F. Cuadrado, "Model-based context-aware deployment of distributed systems," in *IEEE Communications Magazine*, vol. 47, no. 6, pp. 164-171, June 2009, doi: 10.1109/MCOM.2009.5116815.

https://ieeexplore.ieee.org/document/5116815

Model-based Context-aware Deployment of Distributed Systems

José L. Ruiz, Juan C. Dueñas and Félix Cuadrado
Departamento de Ingeniería Telemática-ETSI Telecomunicación
Universidad Politécnica de Madrid
Avda. Complutense s/n (Madrid)
Tel. +34913366831. Fax. +34913367366
{jlruiz, jcduenas, fcuadrado}@dit.upm.es

Abstract

Software deployment deals with the transition of software assets from production to consumers' sites. Distributed systems are created by combination of multiple software components at runtime, possibly running on different devices over a network, making the problem of deployment harder than in the case of centralized systems. This paper presents a model for the description of software components, distributed systems and their dependencies, and a mechanism that uses them to automate the deployment of software units onto devices. These have been implemented and validated in a case study, the Digital Home, which demonstrated its feasibility.

1. Introduction

Software deployment consists of provisioning developed software products to the consumers' targets. After several deployment operations it becomes hard to keep system consistency, because components manifest complex dependencies at different but interrelated levels (ranging from software elements to hardware pieces). All of them must be properly understood and resolved in order to carry out an appropiate action. Despite this fact, deployment challenges have traditionally been neglected by theorists and left over to systems engineering practitioners.

In addition to these difficulties, the recent surge of dynamic distributed computing systems raises the stakes to a higher level. Service-oriented systems are an outstanding and market-relevant representative of distributed systems. In contrast with centralized approaches that release large, monolithic applications, services are created by runtime combination of multiple software components over a network. Furthermore, changes in software are very frequent. Market needs push software providers in a never ending race to include new features, improve performance and enhance user friendliness. As a consequence, system's components have to be constantly updated, configured and removed. However, service provision must still preserve its quality levels. All in all software deployment issues have an undeniable impact on IT systems' operation and maintenance costs.

This paper presents a model for the description of software applications and distributed deployment targets, and a supporting architecture that uses them to automate software deployment onto servers at the deployment target. We use the term context-aware as the executed operations are adapted to the current status of the deployment target.

2. On Deployment Automation

Distributed, component-based systems are one of the key challenges for software deployment [1]. Regardless of the application domain, component packages (such as Linux RPM files, JEE packages and .NET assemblies are the units which are actually deployed. Nonetheless, at runtime they expose objects, libraries and services. Building applications by means of connecting components promotes software reuse and lessens the required effort for creating applications. The SOA (Service Oriented Architecture) paradigm goes a step further, as it promotes creating new functionality by the dynamic combination of services provided by different stakeholders [2]. SOA adds two additional pieces to the complex puzzle of software deployment 1) dynamic availability and 2) runtime binding.

Thanks to the Internet, services can be offered anywhere, in a very short time and at low cost. However, automating deployment is essential to achieve a mature service market. Manual process support is out of the question in large scale scenarios, such as service provisioning to the residential environments [3]. Not only because it is prone to errors, but also because it is not cost-effective.

Deployment is not only a challenge for large scale distribution systems. Enterprise deployment environments, such as banking and e-commerce, do experience similar problems. Business demands more flexibility and agility to adapt to customer needs. This is pushing a change in the core systems from centralized host-based architectures to distributed environments populated by application servers, database systems and Enterprise Resource Planners (ERP). The deployment architecture must cope with 1) the large amount of service components that must be handled and 2) the complexity of the runtime distributed system.

Unfortunately, automating deployment is not a simple task. Solving dependencies (the relation between a software or system component and others it depends upon) is one of the most important open issues. In addition, the dependency resolution process must be version aware, as version changes too often have an impact on the dependent software. Commercial tools such as IBM Tivoli Provisioning provide enterprise system management solutions. They provide an excellent basis for defining and executing deployment scripts over the environment. However, they operate oblivious to the actual state of the system, offering limited reasoning capabilities over the heterogeneous managed resources.

In addition, getting a context-aware distributed deployment system poses more challenges. Although the term “context-aware” has several meanings [4], in the context of deployment, it is constituted just by the controllable parameters which are relevant to this task (leaving apart, for example, user context adaptation). In [5], the authors describe an automatic deployment and configuration system for CORBA applications over a cluster of available nodes. Component automatic reasoning is achieved by including descriptors defined in SPDF (Simple Prerequisite Description Format). They contain both logical and environmental dependencies, which are processed to select the destination node of the desired service. However, these descriptors lack expressivity, and are unaligned to the information modeling standards. In contrast to that, models provide a powerful mechanism to represent the physical context of the environment (including its topology and state), and the logical context represented by the software dependencies.

3. Model-based Deployment Automation

Different approaches have been applied to automate deployment processes. Talwar [6] classifies them as script-based, object-oriented and model-driven. Scripts were the first attempt to automate deployment, and are a very good solution when there is little variability in the process. A better level of reuse, though at a higher learning cost, can be achieved using object-oriented languages, because they benefit of mechanisms such as inheritance and composition, e.g. it is possible to create a new deployment task by extending an existing one. Model-driven deployment brings new possibilities, as resources, systems and applications can be separately modeled and therefore, more specific and intelligent tools can be developed. Moreover, models provide an abstraction layer over the heterogeneity of the managed system. Our deployment architecture proposal is model-based and builds on the basis of the OMG (Object Management Group) D&C (Deployment and Configuration) [7] and the DMTF (Distributed Management Task Force) CIM (Common Information Model) Application model [8] standards.

For adapting deployment to the target we need models that provide an efficient representation of the target topology, capabilities and available resources. The D&C resource model is object-oriented, simple and flexible. Resources are named entities classified into one or more types. Resource instances model physical artifacts, mainly: nodes, bridges and links. Typecasting allows interpreting and understanding the resources' nature, as well as the way to handle them.

In addition, resources can be parameterized with a collection of properties that can be either static or dynamic. Each property has a name, a value and a kind. The property kind determines the consumption nature of the resource, by means of setting a minimum, a maximum, a capacity limit or requiring certain value. Resource managers compare and handle resources consumption at the deployment target according to property kinds and values. The process of determining the feasibility of carrying out deployment actions by type and property matching can be regarded as a CSP (Constraints Satisfaction Problem).

This model fundamentally covers hardware-related resources such as disk space, physical memory or network interfaces. Because of that, it is not sufficient to provide context-aware deployment. Knowledge of already deployed units must be considered to avoid wasting resources, find suitable updates and detect conflicts between software units. Because of that, we have extended the D&C specification, improving its support for software description (adding software resource types, including versioning information and so on). This way, we use the same resource model for describing both software components and the deployment target. Next section presents the details of the model.

4. The Deployment Units Model

In order to automate deployment tasks software components must be described by additional metadata. We propose an XML deployment descriptor that covers the contextual aspects discussed in previous sections. Its syntax and grammar are specified in both XML Schema and eMOF metamodel.

As basis for the deployment descriptor we have adopted the *CIM Application* model. Figure 1 shows the fundamental entities of the model. *SoftwareProductType* is the unit

of acquisition, which can be broken down into software features. A feature represents the unit of election that is offered to a software consumer, e.g. a TTS (Text-To-Speech) product could be modeled as two features, a core feature with basic functionalities and an extra feature with additional voices. The feature description includes a list of software elements, which are the actual deployment units. Units export logical resources (e.g. services), which are described using the extended D&C model.

Very frequently, deployment units have dependencies. We manage dependencies using two mechanisms 1) specifying other deployment unit or 2) specifying a required resource contained in any deployment unit. Obviously the first option is only used for software dependencies. The second is used to specify software requirements such as the need of a certain API, a specific service implementation and also constraints on other resource types, such as required memory, disk space or minimum bandwidth. The model also supports the definition of first-order logic expressions on dependencies, as well as imposing locality constraints, e.g. some units can't communicate over different processes and thus must be deployed in the same node container.

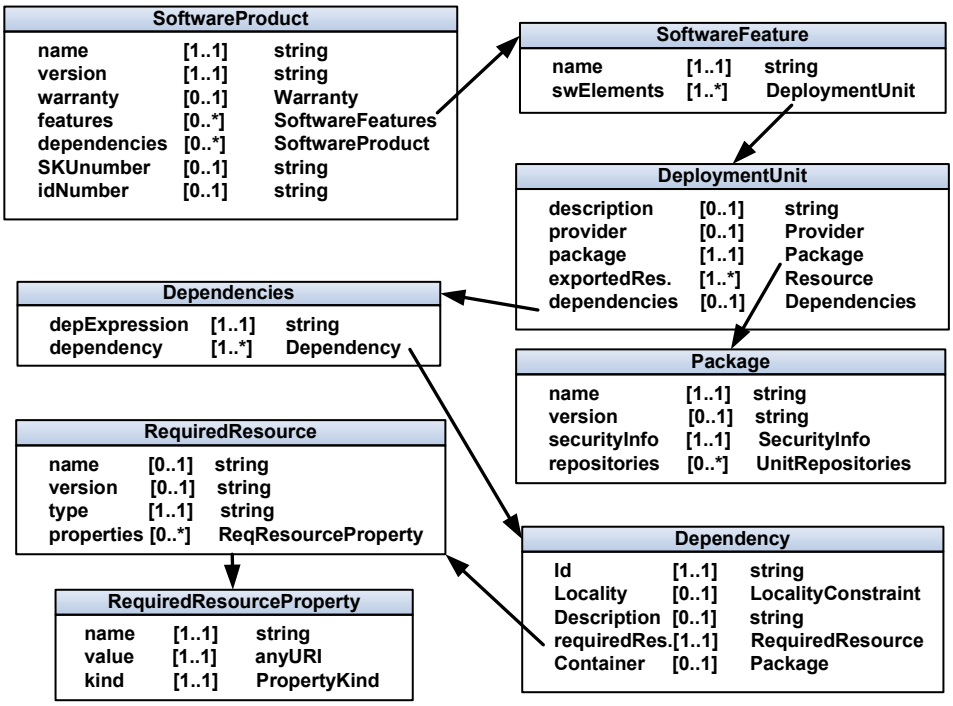

Figure 1 Deployment Units Model

5. Architecture Elements

This section describes the main architecture components that carry out deployment actions based on the models described above. For this purpose we will focus on a specific application scenario, the Digital Home. Figure 2 showcases the two basic entities 1) : active repositories and 2) deployment targets (represented in the figure as peripheral clouds). These must be available in any implementation of the deployment architecture.

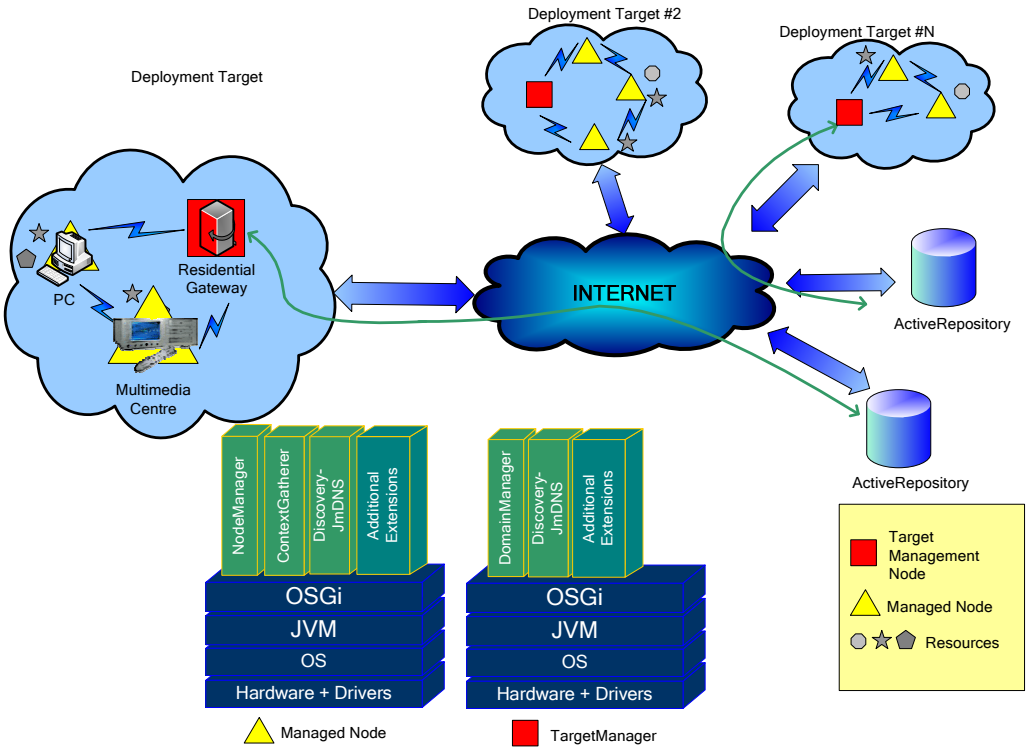

Figure 2 Deployment View of the Digital Home

Each independent management domain constitutes a deployment target. It is composed by one Target Management Node and several managed nodes, which are interconnected through a LAN (Local Area Network). On the left hand side of the picture, managed nodes and the target management node are mapped to real devices in a home target domain. In the Digital Home the centric device is the Residential Gateway (RG). An RG act as a communications gateway and as a Target Management Node for deployment purposes. In the figure, there are two additional managed nodes: a PC and a multimedia centre. The RG can reach one or more Active Repositories over the network.

So far we have just provided a simplified view of the deployment infrastructure. In order to understand the role of the deployment target and the active repository we need to get into details. Figure 3 shows the main components and the most important associations among them. We will start with the deployment target elements and follow with the active repository.

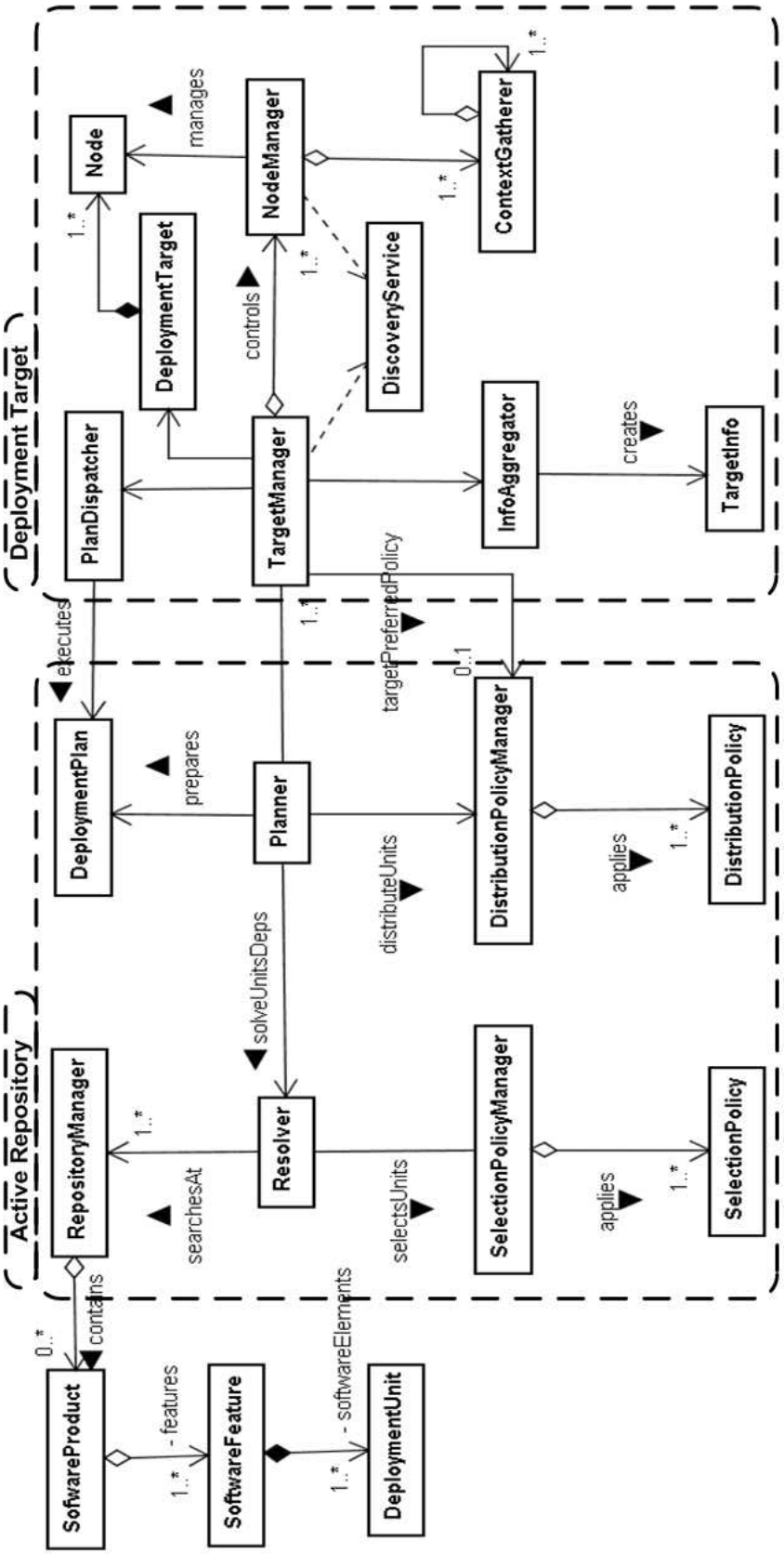

Figure 3 Main Architecture Components

Deployment Target

In the target the ***Context Gatherer*** collects information on the resources of a single node and exposes it through well-defined services using the resource model**.** There can be multiple gatherers providing different kinds of information, such as operating system details (version, name, libraries installed and so on), hardware resources (static capacities and available free resources), deployed units, and so on. This information is aggregated by the ***Node Manager***. The latter executes local deployment operations in its node: installation, activation, deactivation, (re)configuration, removal and update of deployment units.

The ***Target Manager*** coordinates the activities that take place at deployment target level by interacting with active repositories. Its basic responsibilities are 1) providing up to date deployment target information aggregating *NodeManagers* information and 2) the coordination of the deployment plan execution (dispatching actions to *NodeManagers*). For this, the *TargetManager* must be aware of node manager instances running on the target. The target topology could be manually defined by an administrator, and be stored into a configuration database. However, this is usually an arduous task. And even worse, descriptions should be updated according to changes in the target. We have automated this process by means of a ***Discovery Service*** based on DNS-SD (Service Discovery) [9]. Node discovery simplifies operation and maintenance as it is adaptive to deployment target changes.

The implementation of Target and Management nodes for the Digital Home scenario is depicted at the bottom of Figure 2. The target instrumentation runs over an OSGi platform. The OSGi component and service model [10] simplifies the communication between the different agents and allows dynamic binding between the Node Managers and the Context Gatherers.

Active Repository

The active repository 1) registers deployment unit descriptions, 2) supports unit searches, 3) resolves unit dependencies (by providing the name and version of the unit or the specification of required resources), 4) applies policies to select among equivalent units (e.g. multiple implementations of a required service) and 5) selects the distribution of units over the nodes, producing a *Deployment Plan*.

The repository provides CRUD (Create, Read, Update, Delete) operations for deployment unit descriptions. It also provides links to physical packages. These functions are allocated to the ***RepositoryManager.*** In addition, the repository reasons over the models, creates and invokes operations on the target, and thus we call it ‘active repository’.

The ***Resolver*** processes transitive dependencies (expressed in the descriptors) on software resources and deployment units. Each required resource may be satisfied by zero or more deployment units at the repository. If there are several candidates the *Resolver* relies on a *SelectionPolicyManager* to pick one among them. The outcome of the resolution process is a deployment units’ closure that contains the relationships among all the components that must be included in the deployment plan. After this, we need to distribute the unit closure across the deployment target. The *TargetManager* provides information on the available resources and nodes at the deployment target, which is a mandatory input to prepare a plan. Since potentially there are multiple valid

component distributions, a ***Planner*** relies on the *DistributionPolicyManager* to find an appropriate solution. In order to prepare a deployment plan the following information is taken into account:

- Current status of the deployment target: software, hardware and networking resources. Average and static values can also be considered.
- Already available deployment units, so we avoid duplicating units and we can detect potential conflicts between units.
- Deployment units' locality constraints must be checked and ensured.
- End user preferences or business rules could be included as deployment policies, so the process can be executed automatically without human intervention.

The architecture offers two extension points for the provision of policies, each one controlled by its manager: ***SelectionPolicy*** and ***DistributionPolicy.*** Initially we have created programmatic implementations, but we have also performed experiments using linear programming solvers and we devise the application of rule-based servers to contribute that functionality.

All in all, we have provided an agents' infrastructure able to get information about the physical context; defined models for the management of logical dependencies among the software components, and applied a resolution algorithm to select a combination of software components that fit to the physical context and fulfill the logical dependencies. The selection of elements and their allocation to the physical context are automated by the application of policies.

6. Case Study: the Digital Home

We have already mentioned how the Digital Home scenario is an emerging domain where automated deployment can play a fundamental role. In this scenario there is a wide range of services suitable for deployment: multimedia entertainment systems, surveillance services or consumer electronics control, just to mention a few. These services can be accessed through multiple devices, interconnected through local area networks and with indirect Internet access.

At home, the key element is the Residential Gateway (RG). It acts as an intermediary between the existing home networks and devices, and the external networks and service providers. Concerning deployment, the RG plays the role of Target Management Node and cooperates with a remote control centre that includes the Active Repository for carrying out deployment activities. Home devices are considered as managed nodes. The OSGi framework provides a suitable environment for the remote management of digital homes. OSGi was originally designed for embedded systems but it has extended to many other environments ranging from smartphones to application servers. In this scenario deployment operations are carried out over OSGi containers, but we have also applied this approach to Linux Debian packages [1].

In order to illustrate the runtime behaviour of the deployment architecture we are going to focus on a sample case study: the deployment of a multimedia service called ePallantir (http://vlc4osgi.forge.os4os.org/ePallantir/), a multimedia platform created at our research group. Figure 4 shows the product's structure, broken down into two features: the multimedia player and an RSS feed searcher and reader.

The player feature is provided by two deployment units, a player based on the VLC project (http://www.videolan.org/vlc) and a GUI. The FeedSearcher feature is provided by a unit (based on the JDom project http://www.jdom.org/) that provides an API to search in RSS feeds.

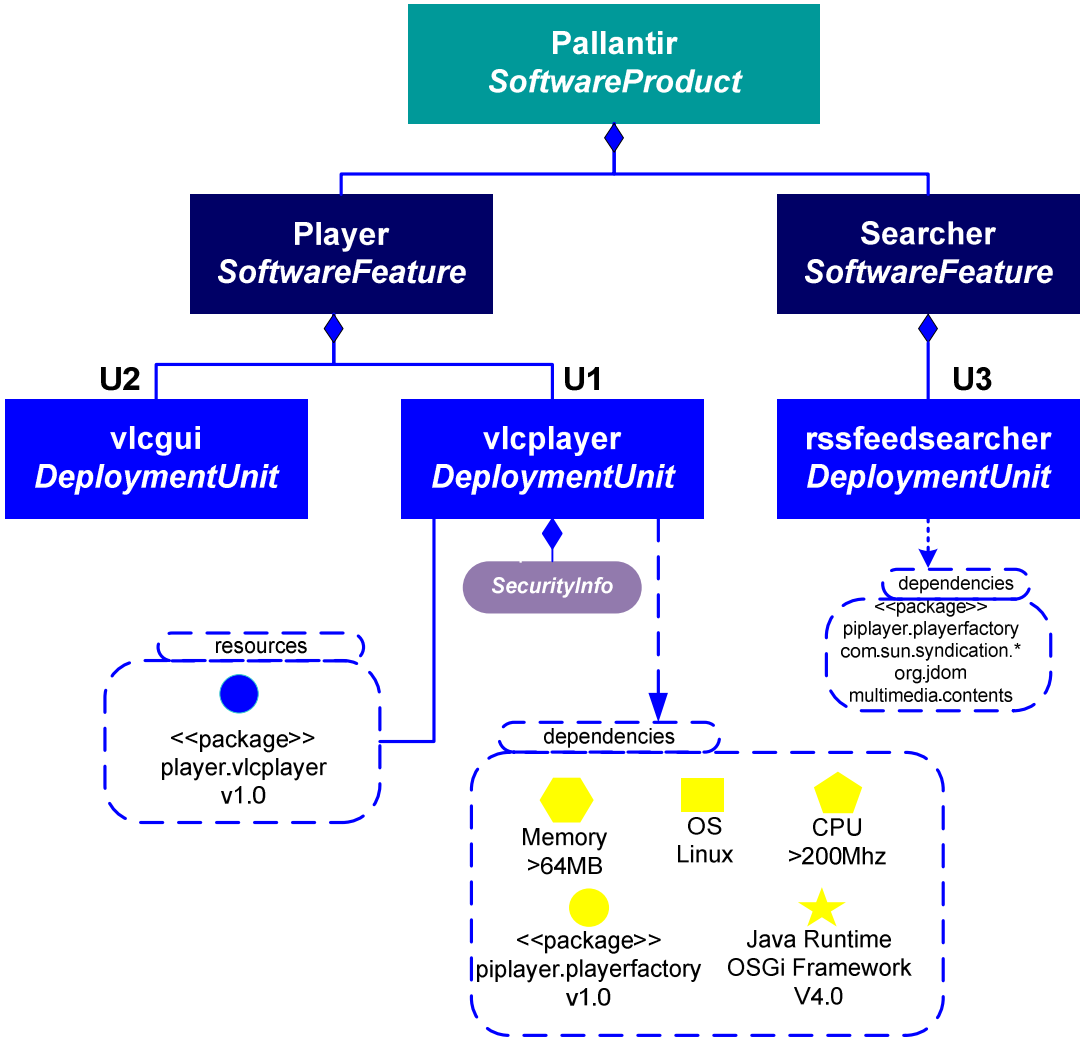

Figure 4 Pallantir: A Software Product Description

Figure 4 depicts the deployment descriptor of the VLCPlayer, which provides the following information (excerpted from the original XML descriptor):

- Dependencies (requirements for the VLCPlayer):
  - Java-Package, multimedia.piplayer.playerfactory, version 1.0.
  - Memory, at least 64 MB RAM available.
  - Operating System, Linux.
  - Java-Runtime, an OSGi framework version 4.0.
  - CPU, at least 200Mhz.
- Exported Resources:
  - Java-Package, multimedia.player.vlcplayer, v1.0.

The deployment target is composed of three nodes: the RG, a multimedia centre and a general purpose PC (as the one appearing at the left-hand side of Figure 2). The RG holds the target manager and the others are managed nodes. At runtime *Node Managers*

gather local information as shown in **Table 1**, and make it dynamically available to the Target Manager.

Table 1 Deployment Target Profile

Node	***Type: HardwarePlatform***		***Type: SoftwarePlatform***	
RG	Memory	Size: 64M	JRE	Version:1.6.0
		FreeNow: 31MB		
	CPU	Architecture: i386	Operating System	Linux Version: 2.6
		AvgLoadLast10min: 0.07		
Multimedia Centre	Memory	Size: 512MB	JVM	Version:1.6.0
		FreeNow: 410MB		
	CPU	Architecture: i386	Operating System	Windows XP Version: Multimedia 10
		Frequency: 1.6Ghz		
		AvgLoadLast10min: 0.23		
General purpose PC	Memory	Size: 2GB	JVM	Version:1.6.0
		FreeNow: 800MB		
	CPU	Architecture: i386	Operating System	Linux Version: 2.6
		AvgLoadLast10min: 0.3		

Figure 5 shows a simplified version of the sequence of operations that take place during the process. In this example the deployment sequence is initiated by the RG, although it can be launched either upon the end user demand or by a remote control centre. The operation requested is the deployment of the Pallantir software product.

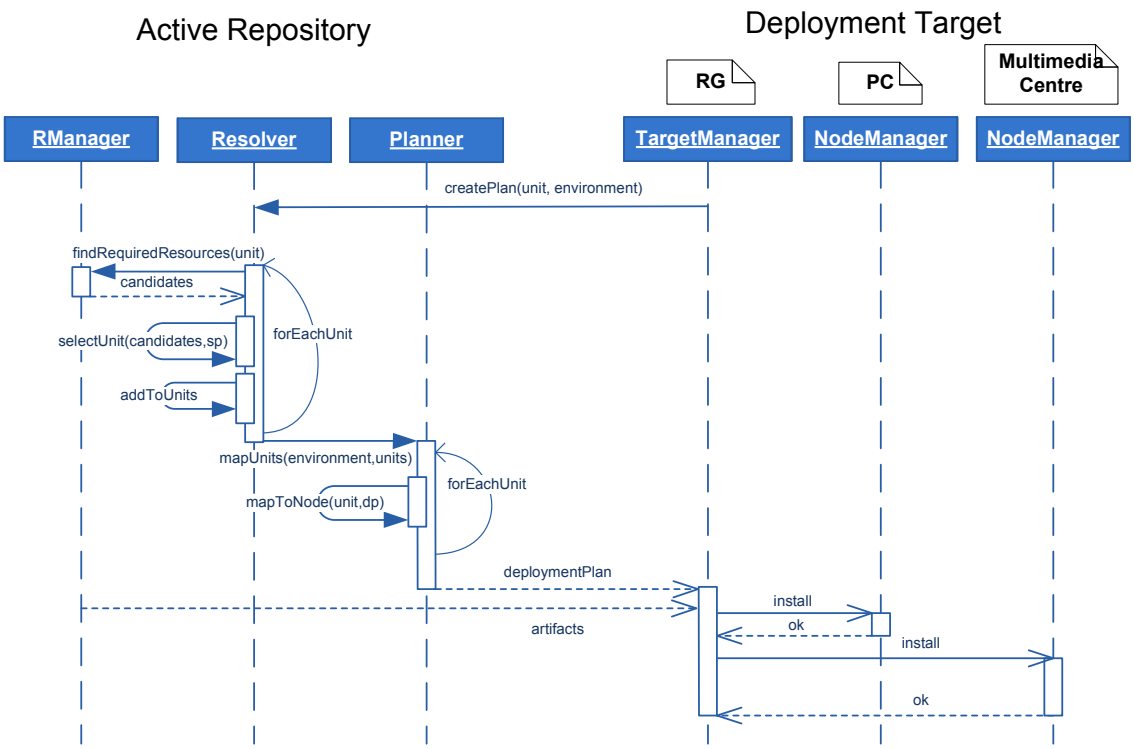

Figure 5 Pallantir Deployment Sequence

First, the Resolver service finds three additional deployment units to satisfy Pallantir's dependencies ending with this unit list:

- U1 (packaged as multimedia.piplayer.vlc4osgi.jar) included in Pallantir's product description. It provides the player component.
- U2 (packaged as multimedia.piplayer.gui_0.8.jar), also included in the original description. It provides the GUI for the player component.
- U3 (packaged as multimedia.ofindu_1.0.jar). It provides the RSS searcher feature of Pallantir.
- U4 (packaged as multimedia_0.9.jar), resolves dependencies on Java packages for U1 and for U3.
- U5 (packaged as org.jdom_1.0.0.jar), solves a dependency on one Java package for U3.
- U6 (packaged as com.sun.syndication_0.8.0.jar), that solves the dependencies on several Java packages for U3.

Once this information is obtained the Planner can elaborate a distribution plan. The *Target Manager* provides information on nodes and resources at home: the Multimedia Centre and the general purpose PC (see Table 1). Then, the Planner generates a plan respecting the hardware resource requirements and the locality constraints (U1, U2 and U4 deployment units must be allocated to the same node, because their dependencies are related to Java package imports). Finally it applies a distribution policy that spreads the load across the domain according to memory sizes.

As a result units U1, U2 and U4 are mapped to the multimedia centre device and U3, U5 and U6 are mapped to the general purpose PC. These two groups can interact over the network, because RSS feeds use the HTTP protocol. The plan is dispatched by the *TargetManager* to the corresponding node managers, which are provisioned with their implementations of the OSGi framework. Node managers install, configure and start units as required by the plan, achieving a valid distributed deployment.

We can see in Figure 5 the main interactions between the Active Repository and the Deployment Target. Uplink traffic (from the target) conveys a "snapshot" of the Target Domain, which accounts for 80Kbytes with no XML payload compression. Downlink traffic contains the deployment units (in this case, 12Mbytes compressed) and the plan operations. After several deployment operations have been performed in a Target Domain, our system will take advantage of it, requiring increasingly less units to deploy a complete application (thanks to the already deployed units).

7. Discussion and conclusions

This paper contributes to the area of software deployment with an architecture that adapts the operations to the current status of the distributed deployment target. Deployment activities can be initiated from remote locations and deployment units can be fetched from multiple unit repositories. Furthermore, the units that collaborate to provide a certain service can be allocated to different nodes at the deployment target preserving their locality constraints.

These goals are supported by the models of both the software components and the distributed deployment target. They include information about the capabilities offered by each element, and the restrictions or dependencies to other elements, usually software components. Software components' constraints must be satisfied by capabilities of the deployment target. Using consistent models for all these elements opens the possibility to automate the processes that: get a current snapshot of the deployment target, calculate a complete set of software components that satisfy dependencies, select among the potential solutions according to policies, identify where in the network each software component will be deployed and finally download and install them. These functions have been assigned in the deployment architecture to the two main entities: the active repository –server side-, and the target managers –clients.

We have performed several proofs of concept for different domains; here we have described the "digital home" case study and the operations required to deploy an advanced multimedia system over a home network. We are doing experiments on deployment of complex environments and business software on server farms, and we are convinced about the usefulness of this approach in automating IT activities.

The home domain does not impose heavy requirements on failure management or performance for the execution of deployment operations. It might happen that the end user initiates a deployment operation while other resource-consuming processes occur at the target, aborting the plan execution. In this case, the operation should be reapplied whenever the resources were available.

Also we understand it is not a closed research, as several aspects have not been considered yet: First, batch operations execution should be supported to implement massive deployment tasks. Also, we intend to improve failure management during deployment plan executions. Ideally, we would like to provide deployment operations with a transactional nature. Improvements in these areas would allow us to apply the concepts in more strict requirements domains.

References

[1] A. Dearle, "Software Deployment, Past, Present and Future", International Conference on Software Engineering 2007, Future Of Software Engineering, pp. 269-284

[2] T. Erl Service-Oriented Architecture. Concepts, Technology and Design. Published by Prentice Hall, 2005.

[3] J. C. Dueñas, J. L. Ruiz, and M. Santillán. *An End-to-End Service Provisioning Scenario for the Residential Environment*. IEEE Communications Magazine, vol. 43, no. 9, September 2005.

[4] D. Ayed, C. Taconet and G. Bernard *A Data Model for Context-Aware Deployment of Component-based Applications onto Distributed Systems*. Component-oriented approaches to context-aware systems Workshop ECOOP 2004.

[5] F. Kon, J.R. Marques, T.Yamane, R.H. Campbell, M.Micunas,"Design, implementation, and performance of an automatic configuration service for distributed component systems", Software Practice and Experience, 2005, 35, 7, 667-703

[6] V. Talwar, D. Milojicic, C. Qinyi Wu Pu, W. Yan and G. Jung. *Approaches for service deployment*. IEEE Internet Computing Magazine, 2005, vol. 9, no. 5, pages 70-80.

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

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

[9] M Krochmal, S Cheshire, *DNS-based Service Discovery,* Internet Draft, August 2006, http://files.dns-sd.org/draft-cheshire-dnsext-dns-sd.txt

[10] Open Services Gateway Initiative. *OSGi Service Platform,* Specification Release 4.1,May , 2007.