

Towards automated software component configuration and deployment

Vincent Lestideau, Nouredine Belkhatir, Pierre-Yves Cunin

Adele Team Bat C
LSR-IMAG , 220 rue de la chimie
Domaine Universitaire, BP 53
38041 Grenoble Cedex 9 France
E-mail: {Vincent.Lestideau, Nouredine.Belkhatir, Pierre-Yves.Cunin}@imag.fr

Abstract

Software deployment is a complex process that covers post development activities as configuring, releasing, installing, updating, adapting until uninstalling a software application. The existing tools do not cover all this process and they are ad hoc. This paper presents and describes some ideas to create a deployment environment. The aim of this environment is to ensure a coherent and secure installation. We propose an approach using the federation technology.

Keywords: *Deployment, Configuration, Component Model, Deployment Tool, Federation.*

1. Introduction

Software deployment is a complex process that covers post development activities as configuring, releasing, installing, updating, adapting until uninstalling a software application. The development of component-based applications has significantly increased the importance of the deployment process. Recently new technologies have emerged to address the deployment problems. The deployment tools are evolved, they cover more and more of the deployment spectrum. In parallel, the new component model specifications take into account the deployment. Now the deployment is partially realised during the development and assembly steps. This newest approach allows an improvement in the quality and the possibilities of standard deployment tools.

In this paper, we present the deployment throughout the component model specifications and the existing deployment tools. Despite these

improvements, we have not found an approach covering all the deployment processes. The aim of our paper is to propose a deployment environment, which take into account all the tools and component programming. We provide a federation approach to federate these deployment tools. The paper is organized as follows. In the next section, we present the software deployment process. Section 3 presents some deployment tools and two new component model specification views from the deployment angle. In the section 4, we describe our approach and we conclude with some perspectives about future research.

2. Software Deployment Process

According to Carzaniga and al. [1], software deployment process (figure 1) refers to all the interrelated activities, which represent the life-cycle of an application. These activities are related to the release, install, de-install, update, adaptation, activation and deactivation of software. In this paper, we are more interested in the release and install activities, and more particularly, in the case of component-based software, other activities are briefly presented.

Activate and De-Activate. These activities consist to startup and shutdown the software components. That concerns all necessary steps to properly run (or shutdown) an application.

Update. It is a special case of the install activity. It consists of installing another version of an application already installed. It is less complex than the install because many of the needed components are already installed.

Adapt. Like the update activity, the adaptation involves to modify a software already installed. The same release is kept but only modified in terms of configuration.

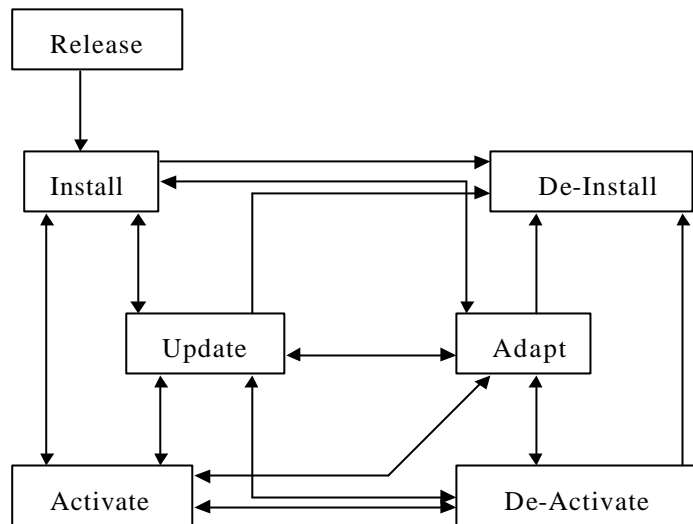


Figure 1 : Software Deployment Process

De-Installation. It consists of removing an installed application. This removal must be done properly. Many difficulties are related to shared components and the registry update.

Figure 2 shows a view of the release and installation activities. We added the configure activity, which allows to create the different packages corresponding to a release.

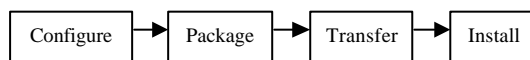


Figure 2 : Configure and install activities of the deployment process

Configure. One interesting aspect of component-based software is the possibility of reuse existing components to create new applications. Generally, available components are stored in a component development base, and each of them may have several versions. In section 4.3, we explain our selection process that consists of annotating the different components and selecting those corresponding to a needed configuration.

Package. It consists of recovering and wrapping the components issued from the configure activity and their annotations. A package may also contain a set of software dependencies (necessary to run the application), some deployment information (package number, creation date,...) and a description of the packaged application (a manifest). This description will be used to determine the good package to be installed in a particularly site, with respect of its configuration and constraints.

Transfer. This activity means the transport of the package from the producer to the consumer site. The more important difficulties are related to firewalls and network bandwidth.

Install. It is the activity of unpacking the received package, installing the different components, modifying the registry base, resolving the

dependencies, and in some cases, running another installation related to a depending package.

The deployment is an important stage in software engineering. Several existing tools try to cover all the deployment process. Component model specifications focus the deployment issues. In the following section, we present some existing deployment tools like InstallShield Developer, Java Web Start..., and some component model specifications like CCM, .NET.

3. Related Work and Tools

When the deployment process takes an important place in software engineering, a large number of related tools have been developed. This section is dedicated to present two related tools and two related specifications of the deployment process.

3.1 Deployment Tools

Many existing tools does not support all the deployment process (figure 1), but only one or several activities. In this section, we describe succinctly two tools

3.1.1 InstallShield Developer

Today, InstallShield Developer [2] is the most known installer. It applies a process composed of three different activities: creation, distribution and maintain of applications. InstallShield Developer is a specialised tool for windows platform-based applications. It uses all Microsoft's Windows Installer service [3] features. This service provides automatic means to repair the corrupted files of an application, allows to properly install and de-install applications, and offers a rollback service to return to a computer's original state after a failed installation. To use this service, InstallShield Developer creates an MSI file, according to Windows Installer Format. This file contains some package information (such as version, date or

vendor's name), a relational database of installation instructions, containing the registry entries, shortcuts, etc. and the application files or references to them (if they are stored outside the package).

The creation activity specifies the application's files, registry entries, and resolves the dependencies. At least, a Windows Installer Package is created. The distribute activity refers to the package transport. Different means are provided: via the web, via an intranet, via an installation CD... Once on the physical site, the package is read and run by the Windows Installer service. The maintain activity consists of updating the package. InstallShield Developer allows to write some patches to avoid rewriting all the package. The application consistency is ensured by Windows Installer service.

In terms of deployment, InstallShield Developer provides an efficient solution, in the case of Windows applications. This tool improves the installation and the coherency of the installed applications.

3.1.2 Java Web Start

Java Web Start [4] is an application deployment solution for the Web. It deploys Java-technology-based applications from either the Web or the Java Application Manager. A simple click on a Web page launches the application. If the application is not present on the target machine (the application is installed for the first time), Java Web Start automatically downloads all necessary files. It then caches the files on the target machine, so the application is always ready to be relaunched anytime the user wants. Before the application is launched, Java Web Start checks if a more recent version exists on the server, therefore, only the newest application version is executed on the target machine.

In terms of deployment, Java Web Start provides a solution allowing to execute always the last application version. This system can resolve the update problems and maintain the system consistency, but this implies it to be regularly connected to a server.

3.1.3 Synthesis

A large existing number of tools proposes partial solutions for deployment problems. In some cases, they are very efficient, but do not cover all the software deployment process. Some of them (like Java Web Start) cover almost all of the deployment spectrum, but the applied constraints are very important (Java application, web server...).

3.2 Component Model Specifications

Recently, some new component model specifications try to address explicitly the deployment issues. In this section, we describe briefly two different approaches trying to resolve

some deployment problems like versioning and software dependencies.

3.2.1 Corba Component Model (CCM)

CCM, is a component specification developed by the OMG [5]. It is based on the concepts of CORBA distributed architecture. CCM explicitly specifies a component deployment process, some component deployment objects and a deployment scenario. In CCM, the components are packaged into a self-descriptive package, which can be assembled into assemblies that can be deployed. To perform the CCM applications deployment, it is necessary to specifically define the list of the component instances belonging to the application, their logical locations and their connections. For a component, it is necessary to specify its elements (interfaces, implementations), to describe the system requirements (OS, ORB...) and the initial configuration. All the produced data is necessary to install and run a CCM application. CCM specifies some package descriptors, such as software package descriptor, component descriptor and assembly descriptor. These packages are based on the OSD [6] specification. Using these different packages and their descriptors, CCM specification proposes a deployment scenario to instigate the proper installation of the CCM application.

In CCM, the deployment is performed via the available ORB services and deployment tools. These tools use the ORB services to perform component transfer, installation, assembling, instantiation and the configuration on the targeted sites.

3.2.2 Microsoft .NET

.NET Framework [7] is Microsoft's answer to resolve the versioning and deployment problems due to the DLLs. Known as DLL hell [8], the versioning problem appears, for example, when a new application installs a new version of a shared component that is not backward compatible with the version already installed on the machine. Microsoft introduces a new concept called assembly. Assemblies are the entities manipulated by the Framework, they are composed by the code of the components, their resources and a manifest. The manifest includes some identity data, the file list, the dependencies and other data used to execute the code. The .NET framework implies some constraints:

- the assemblies must be self-describing; so the installation has no impact on the registry, therefore the installation and de-installation activities are more simplified.
- the assemblies must respect a version number policy; this policy can be customised.
- the framework supports the side-by-side components allowing multiple versions of a component to be simultaneously executed on the machine.

- the applications must be isolated as much as possible; an assembly can only be accessed by one application. The shared assemblies are supported by adding some requirements such as global and unique names.

In .NET, the problem of versioning during the installation seems to be resolved, but some constraints like the 'no-shared' assemblies can be difficult to ensure. In terms of deployment, .NET improves the quality of the installation by decreasing the risks of modifying the already installed application that share some components.

3.2.3 Synthesis

By specifying the deployment requirements, the presented specifications improve the deployment activities, because the deployment constraints (in terms of dependencies, components physical localisation...) are resolved in advance. Another advantage of these specifications is that the deployment tools can use these standardised descriptions. The tools become generic and can be used to deploy some component-based applications.

3.3 Sum up

It appears that the deployment issues are more and more addressed either by some deployment tools and component specifications. The tools try to resolve and automate the problems of installation and maintaining in particular. Specifications which try to help these deployment tools by resolving some problems before the installation. In the next section, we propose one approach, which allows to take advantage of all these solutions, by using tools in the case of component-based deployment.

4. Our Approach

4.1 Deployment Environment

As seen previously, existing tools does not cover all the software deployment process. On the other hand, a wide variety of tools supports subparts of this process. Our approach consists of using these tools to constitute a deployment environment. These tools are not developed to work together and they can not be modified, thus we have decided to use the federation technology, developed in our team [9].

Federation technology permits to create some applications by federate some existing tools. These tools are designed to be used standalone (i.e. they are independent of other tools), they can not be modified and they can interact with the environment. We do not explain in this paper the means used to federate the tools together. Thus all the deployment process can be automated and lead by the federation (by the intermediate of a process engine named APEL [10] developed in our team). A federation is described in terms of roles which are

the only entities manipulated by the federation, the tools (playing the roles) are not considered by the federation. To use them, we must create some wrappers. This technique permits us to be independent of the tools and we can replace a tool by another, without problems.

Today, we have developed the configuration and installation parts of our deployment environment. We have implemented our own selection tool based on the selection process defined in section 4.3. This tool uses a component base to create a software package respecting a configuration. Furthermore these packages may be used by installer tools such as InstallShield Developer, and after, the application is properly installed by other tools, described in section 4.4.

4.2 Our Component Model

Our component model [11] is based of the Corba Component Model which introduced a new concept called externalisation. A component composed by other components (a composite) is also considered as a component.

Our component model is described in figure 3. The first activity of our software deployment process is the software configuration. We have introduced in our model an intermediate level in the component composition (represented by the *refined relation*). We have explicitly taken into account the binding by defining some composite components partially defined.

To perform the selection, we have described all the component entities with an annotation. An annotation object is composed of a set of attributes. It describes the component characteristics according to a particular life cycle phase (the deployment phase in the context of this paper). Attributes are typified and multi-valued. For instance, an annotation may describe a component in terms of equipment constraints (processor) or software constraints (language, operating system). Our components are hierarchical (an interface is implemented by an implementation, which can be refined in the case of composite implementation) and the annotation must take into account this hierarchy. For instance, an interface conceived to a Unix platform must be implemented by some implementations, which respect this constraint. And we must retrieve this relation at the associated annotations level (for instance with the attribute "*Operating System*" and the value "*Unix*").

By a system of propagation and rules (not presented in this paper), we assure the consistency of our annotations. This consistency is important to realise a selection process, and to obtain a valid configuration.

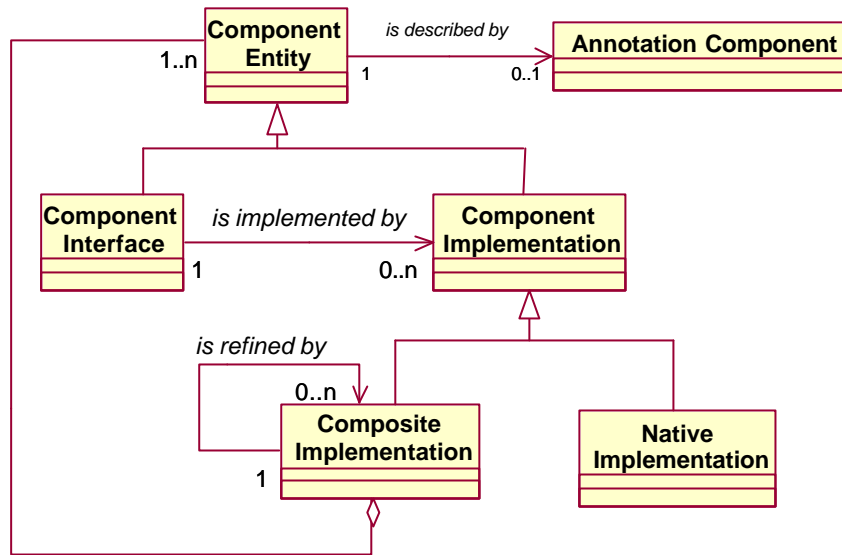


Figure 3: the component model

4.3 Our Selection Process

Interfaces constitute the root elements in the component development base. This base contains all the component entities (interfaces and their implementations) which the selection is made. Each interface may be the root of many implementations tree. The main aim of the selection process is to cross down the component database and to prune all branches that do not match with the configuration request. This can be done by taking advantage of annotations (see before).

To perform the selection, a state is associated to each component. This state may take one of three values: *selected*, *selectable* or *not-selected*. The principle is to keep only components having a selected state. To calculate the component state, the first step is to verify if the component matches the request. In this case, its state takes selectable value. Otherwise it takes the not-selected value and the component branch is pruned and a backtrack is performed.

If the component is selectable, the following sub-process is applied to check if the component can be finally selected. This sub-process is dependent on the component type (interface, basic component or composite component):

- An interface becomes selected if it does not have any children (implementations) or if at least one of its children (basic or composite) becomes selected.
- A basic component becomes automatically selected because it does not have any children.
- A composite component that is a leaf for the component family becomes selected if and only if all its occurrences are selected. Refined composite becomes selected if at least one of its children becomes selected.

During this process, the component annotations are used to check if the component is consistent with the request. In a positive case, the component becomes selectable. The consistency is in part ensured by the use of annotations.

At the end of the selection process, we dispose of a set of components and their annotations. We create an annotation describing the software using the component annotations. The component package contains also components dependencies. It will be installed thanks to our installation federation.

4.4 Our Installation Federation

The federation is constituted of a set of tools. In terms of architecture, we have identified three entities; (1) a set of application servers, (2) a deployment server and (3) a set of physical sites. An application server contains available packages ready to be installed on the physical sites. A package is the result of an installer tool like InstallShield. We added a description in the packaged software, used to determine a package according to the target site constraints and corresponding to the wished configuration. A physical site contains a site model describing its configuration and the applications already installed. The deployment server contains the federation engine.

To automate the installation process, all available packages must implement an installation API, providing a set of operations such as installation, de-installation and other maintaining operations, without have knowledge of the technology used by the different installer tools. For instance, in the case of InstallShield, we have just to write a method running the executable file that launch the installation.

In our installation scenario, we have identified 3 roles: *installer*, *transfer*, *analyser*. The installer role creates an executable performing the

installation (like InstallShield Developer), the transfer role is a tool allowing to transfer a package throughout a network (even the target site is behind a firewall). The analyser role is a tool which analyses the consequences of an installation on a physical site (in terms of shared components and version problems). Another advantage of the federation technology is to propose a communication infrastructure which ensure the file transfer even in the case of firewalls. The only disadvantage is that all communications must transit by the deployment server.

In the purpose to create a deployment federation that covers all the software deployment process, this installation federation represents the kernel around it other roles and entities will be added.

5. Conclusion and Perspectives

The research goal in our study is to improve the deployment support leading to zero administration that will reduce efforts. In part this goal may be reached by formal models (applications, sites , deployment process) and with the help of internet and intranet connections using a distance application servers. One of the major insights gained from this study and contribution is the definition of a deployment framework. There are many more deployment strategies that can be derived from this framework. Having defined the deployment framework and some prototypes, we are now concentrating on validating our approach.

References

- [1] Carzaniga, A. Fuggetta, R.S. Hall, A. van der Hoek, D. Heimbigner, A.L. Wolf. A Characterization Framework for Software Deployment Technologies. Technical Report CU-CS-857-98, Dept. of Computer Science, University of Colorado, April 1998.
- [2] Web site InstallShield Developer
<http://www.installshield.com/isd/>
- [3] Web site Microsoft's Windows Installer
<http://www.microsoft.com/>
- [4] Web site Java Web Start
<http://java.sun.com/products/javawebstart/>
- [5] J. Mischkinisky - *CORBA 3.0 New Components Chapters* - CCMFTFDraft ptc/99-10-04 -- OMG TC Document ptc/99-10-04 October 29, 1999
- [6] A. van Hoff, I. Hadi Partovi, T. Thai - *The Open Software Description Format (OSD)* – Site Web : <http://www.w3.org/TR/NOTE-OSD>
- [7] Web site .NET Framework
<http://msdn.microsoft.com/library/>
- [8] Rick Anderson - The End of DLL Hell- Microsoft Corporation – January 2000
<http://msdn.microsoft.com/library/>
- [9] Jacky Estublier and Anh-Tuyet LE. Design and development of Software Federation ICSSEA 2001. Paris, France. 4-6 December 2001.
<http://www-adele.imag.fr/Les.Publications/intConferences/ICSSEA2001Est.pdf>
- [10] S. Dami, J. Estublier and M. Amieur. APEL: A Graphical yet Executable Formalism for Process Modeling. Kluwer Academic Publishers, Boston. Process Technology. Edited by E. Di Nitto and A. Fuggetta..Pages 61 - 97. January 1998.
- [11] N.Belkhatir, P.Y. Cunin, V. Lestideau and H. Sali.- *An OO framework for configuration of deployable large component based Software Products*- OOPSLA 2001 Workshop on Engineering Complex O.O. systems for Evolution. Tempa, Florida, USA. 14-18 October 2001