

| | |
|--------------|---|
| Title | ミドルウェアシステムにおけるソフトウェア構成管理に関する研究 |
| Author(s) | Pimruang, Adirake |
| Citation | |
| Issue Date | 2004-09 |
| Type | Thesis or Dissertation |
| Text version | author |
| URL | http://hdl.handle.net/10119/1886 |
| Rights | |
| Description | Supervisor:落水 浩一郎, 情報科学研究科, 修士 |

Research on Software Configuration Management in Middleware Systems

By Pimruang Adirake (210201)

A thesis submitted to
School of Information Science,
Japan Advanced Institute of Science and Technology,
in partial fulfillment of the requirements
for the degree of
Master of Information Science
Graduate Program in Information Science

Written under the direction of
Professor Koichiro Ochimizu

and approved by
Professor Koichiro Ochimizu
Professor Takuya Katayama
Associate Professor Masato Suzuki

August, 2004 (Submitted)

Abstract

From the widely use of the component middleware, developers can reuse existing components developed by in-house development or provided by other organizations. Each organization can develop components independently in different areas. Some components in an organization can be provided via the Internet to deployment processes of other organizations. Developers need to handle versions of interrelated components in both of development and deployment. J2DEP is a system to generate and manage component dependencies in development process, and apply them to automate deployment. The key feature of J2DEP is to manage the dependency information between in-house components and third vendor components. Then, J2DEP can manage the dependency information in the connection with configuration management system. It uses the dependency information in release and deployment to provide consistency set of components.

Contents

| | |
|--|-----|
| Contents | i |
| Figure Lists | iii |
| Chapter 1 Introduction | 1 |
| Chapter 2 Background | 3 |
| 2.1 Component Development-Deployment..... | 3 |
| 2.1.1 Application Component Provider..... | 3 |
| 2.1.2 Application Assembler..... | 4 |
| 2.1.3 Deployer..... | 4 |
| 2.2 Software Configuration Management | 4 |
| 2.3 Software Deployment..... | 5 |
| Chapter 3 Example Scenario | 6 |
| Chapter 4 Our Approach | 8 |
| 4.1 Component Development Support..... | 10 |
| 4.2 User Deployment Agent..... | 10 |
| Chapter 5 Dependency Metadata | 11 |
| 5.1 Metadata about component details..... | 11 |
| 5.2 Metadata of Dependency component..... | 12 |
| 5.2.1 Metadata of Dependency component from source repository..... | 12 |
| 5.2.2 Metadata of Dependency component from release site..... | 13 |
| 5.3 Dependency Metadata of component from the other development systems..... | 14 |
| Chapter 6 Implementation | 16 |
| 6.1 Development Space Structure..... | 16 |
| 6.2 Development Space Initialization..... | 17 |
| 6.3 Dependency Component Import Tool..... | 18 |
| 6.4 Component Packaging and Developer Deployment Support..... | 20 |
| 6.5 Component Release Tool..... | 20 |
| 6.6 End-User Deployment Agent..... | 21 |
| Chapter 7 J2DEP Development-Deployment Process Example | 22 |
| 7.1 Development process by J2DEP..... | 22 |
| 7.2 Deployment process by J2DEP..... | 24 |

| | |
|---|----|
| Chapter 8 Related Works | 26 |
| 8.1 Eclipse, NetBeans, JBuilder..... | 26 |
| 8.2 The RPM Package Manager (RPM) | 26 |
| 8.3 Software Release Management (SRM) | 26 |
| 8.4 Two-Way Integrated Configuration management and Software deployment (TWICS)..... | 27 |
| Chapter 9 Conclusion | 30 |
| Acknowledgements | 31 |
| References | 32 |
| Appendix | 33 |

Figure Lists

| | |
|---|----|
| Figure 2.1: The role of each tool in development-deployment flow..... | 5 |
| Figure 3.1 Example Scenario of Component-based development..... | 6 |
| Figure 4.1 Example of Development space..... | 8 |
| Figure 4.2 J2DEP Architecture..... | 9 |
| Figure 5.1 Metadata about component details..... | 11 |
| Figure 5.2 Metadata of Dependency component from source repository..... | 13 |
| Figure 5.3 Metadata of Dependency component from release site..... | 13 |
| Figure 5.4 Dependency Metadata of component from the other system..... | 14 |
| Figure 5.5 Example of dependency component from the other system..... | 14 |
| Figure 6.1 J2DEP Implementation..... | 16 |
| Figure 6.2 Development space, Source repository and Release site..... | 17 |
| Figure 6.3 Component dependencies in development space..... | 18 |
| Figure 6.4 Import tool for Binary Component from release site..... | 18 |
| Figure 6.5 Import tool for Component Source from repository..... | 18 |
| Figure 6.6 Component Release Tool..... | 21 |
| Figure 7.1 J2DEP Development Support Step 1 and 2 | 22 |
| Figure 7.2 J2DEP Development Support Step 3 and 4..... | 23 |
| Figure 7.3 J2DEP Deployment process in developer middleware..... | 24 |
| Figure 7.4 J2DEP Release process..... | 24 |
| Figure 7.5 J2DEP Deployment on user middleware Step 1 and 2..... | 25 |
| Figure 7.6 J2DEP Deployment on user middleware Step 3 and 4..... | 25 |
| Figure 8.1: Functionalities of J2DEP and related tools..... | 28 |
| Figure 8.2: Comparing each tool on Development-Deployment process flow..... | 29 |

Chapter 1

Introduction

The middleware, architecture for the development and deployment of software components, is now widely used in business (e.g., J2EE[1,2], .NET[3], CORBA[4]). Each component is an encapsulated part of a software system implementing a specific service or a set of services to support business requirements. To build large business systems, developers need several functionalities from the existing components. Each organization can reuse their own in-house developed components or purchase components from third-party vendors to construct the system in middleware technology. The reuse of existing components in middleware can reduce the time and cost of development [5].

Each component can be provided by in-house development or come from other organizations distributed in different locations. Such organizations generally publish their components to their release sites as binary units to avoid source code release [6]. The component provider companies can integrate binary components to develop new components or applications [7]. Developers in such companies could not acquire the dependency information of third vendor components when they want to integrate these components into a new component or an application. They need to resolve component dependencies manually when they adopt the third vendor's components. In deployment phase, it is also troublesome and consumes time to deploy the proper version of component without component relationship information. Another problem may occur to end-users when the version of a component is changed. Version conflicts may appear in the deployment phase because the effect of the change can not be traced [16].

Current Software Configuration Managements (SCM) do not well provide to improve and enable component-based development and reuse [8]. Different organizations can provide several version of components in which their relationships are not explicitly described [9]. These systems can not manage the evolution of components developed by third-party organization properly. Each third party organization develops his components independently and releases them in different release policy. For example, same developers may release their components' sources in repository and release binary component in HTTP server. These systems can not manage the dependency information based on different release policies.

In additions, the software deployment functionality, following activities: packaging, releasing, obtaining component and dependencies, should be done in automatical ways [10]. To support automatic deployment process, developers generally have to define dependencies

in release phase. Then, deployers use the deployment tool to obtain the component defined in dependency information from the release site. However, in component-based development, the process of component identification starts from design phase[11], so, developers want to use this dependency information in development phase rather than redefine it again in release phase.

What we need is the configuration management system that supports component-based development. Configuration management must support managing component relationship and the change of component versions by different organizations. This means such a system must help developers to adopt components based on different release policy, help developers to generate component dependency in development phase and manage both source and dependency information together in SCM repository. With this system, deployment phase can be performed automatically. Developers can use the dependency of components to obtain the correct version of components to be assembled and deployed in developers' middleware in build and test phase. Also, deployers can get correct version of components to deploy in user middleware in component deployment phase.

In this paper, we propose J2DEP (J2EE DEvelopment-dePloyment support) which supports configuration management addressing both in development and deployment phase. J2DEP helps developers to create or generate dependency information from imported components and manages source and dependency information inside a CVS repository to support development process. Moreover, J2DEP can also publish the binary components to release sites by using FTP/HTTP server. In the development process, J2DEP help developers to import remote components, developed by different organizations, to its development environment by getting source files from the repository or binary component from the release site. Then, it generates the *dependency metadata* from imported components and control metadata files in the repository. The dependency metadata mainly represents the component detail (e.g. name, version type) and the method to obtain the component from repositories or release sites. Finally, in deployment phase, J2DEP helps developers to get component from release site, to assemble relating components into application components and to install component and dependency to the target middleware.

The paper is further structured as follows. We show the background of this research in chapter 2 and give an example scenario that motivates this research in chapter 3. We brief the overview and approach of J2DEP system in chapter 4. Dependency metadata is described in chapter 5. In chapter 6, we give the implementation details of J2DEP system both in development and deployment phase support, and an example of process flow with J2DEP in chapter 7. Chapter 8 shows the related work. Finally, we show some conclusion in section 9.

Chapter 2

Background

J2DEP intends to support both component development and deployment phases. In this section, we would like to discuss about related work and motivation of this research.

2.1 Component Development-Deployment

To realize the problem raised in development process in middleware, we would like to show the development roles and tasks in J2EE. J2EE Development-deployment roles [2] are consists of 3 main roles following: (1) Application Component Provider (2) Application Assembler (3) Deployer

2.1.1 Application Component Provider

Application Component Provider provides the building blocks of J2EE application. Developer can develop a component and package a binary file into application component. Component from application component provider may have the dependency on the other component. In J2EE, there are 2 methods to handle the dependency component.

2.1.1.1 Package dependency component into a new component:

The dependency can be reduced by grouping the related component into a new component. However, this method reduces the degree of component reusing because dependency component has been a part of the new component.

2.1.1.2 Do not package dependency component:

To maximize the reusability of each component, we have to leave a room for the Application Assembler to pick and select components to compose J2EE application

The problem raised in Application Component Provider role is about the dependency of components. In development phase, developer can not get dependencies from configuration

management system. When developers check out component from a source repository, developer should obtain both source and dependency. TWICS[4] can use metadata defined in release phase to get dependency. However, dependency metadata in development phase may be reduced in release phase because some components may be a part of another component. So, components in repository and release site may have different dependency metadata.

2.1.2 Application Assembler

Application Assembler role is to group the set of components developed by application component providers and assembles them into J2EE application. An application assembler is responsible for providing assembly instructions describing external dependencies of the application that the deployer must resolve in deployment phase.

To support application assembler automatically, the developer need to use dependency metadata to obtain related component and to generate deployment descriptor about external dependencies.

2.1.3 Deployer

A deployer installs components and applications into a J2EE server and configures components and applications to resolve all the external dependencies declared by the application component provider and the application assembler.

By development role in J2EE, a new component may have dependencies on the other components, so Application Component Provider should define dependency properly in development phase because Application Assembler and Deployer need it to resolve the dependency. In development phase, since the dependency can be obtain from repositories or release servers, developers should get a dependency from different repositories or release server automatically.

2.2 Software Configuration Management (SCM)

Software configuration management concept is to manage the charge of software artifacts in development space. The most of SCMs including RCS[12] and CVS[13] can manage only text file. While component-based development, developer have to deal with both component source in text format and binary component. So, we need a method to manage the change in binary components on local SCMs when developers want to reuse them.

2.3 Software Deployment

The software deployment life cycle is evolving these activities: package, release, configure, assembler, install, update, remove, adapt [14]. The most of deployment tools can support component development and can manipulate the component dependencies. Some tools can support component development by distributed organization. Also, SRM[15], Software Dock[14], RPM[17] (Figure 2.1 shows comparison of each tool) can manage multiple version of component. But these tools do not connect the deployment process with SCM. TWICS[4] resolves this shortcoming by supporting to get components from configuration management repository (third vendor repository or in-house repository) to release and get the component from component publisher to local SCM repository (source repository of in-house development).

However, deployment tools we mentioned above seem not to connect development phase and deployment phase properly. The component dependencies are defined in release phase but, by reuse concept, related components have been defined in design phase. The developer need to use dependency defined in deployment phase especially in build and test phases. So dependency should be defined early in development phase rather than in release phase.

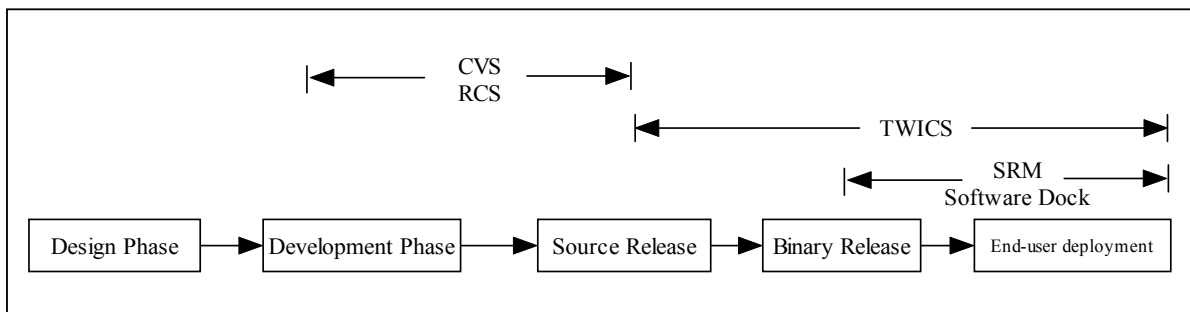


Figure 2.1: The role of each tool in development-deployment flow

Chapter 3

Example Scenario and Motivation

To clarify the issues of component development-deployment in middleware, we consider relations of components developed by different organizations shown in Figure 3.1. Rectangle boxes represent component developed by each organization displayed in oval shapes. The arrows show the dependencies among the components. Text below the rectangle boxes show the component name, version and release type respectively.

In Figure 3.1, component AccountManager developed by organization DCom is depended on component DbConnector from organization DataNet. Also, component UserService developed by organization ATech is depended on component AccountManager and component UserTransaction from organization DCom and component ProfileFormat from organization CyberC. Some component is developed by only one organization and some component can be developed by cooperative of different organizations. For example, DCom originally develop UserTransaction, but DCom can share source of UserTransaction to ATech to support component UserService development.

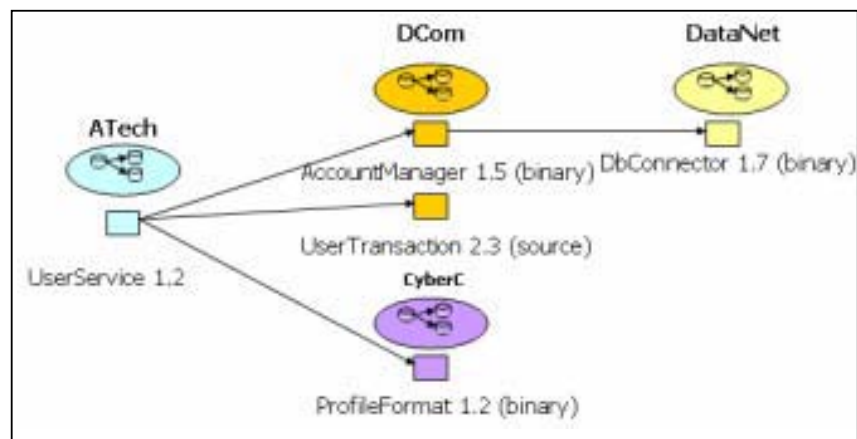


Figure 3.1: Example Scenario of Component-based development

First issue is each organization may develop several versions of components and each component may depend on the other components. Developers have no support to document the dependency information properly. They need to resolve dependencies manually whenever they

get the source from repository to the development environment. For example, the dependencies of UserService 1.2, which are AccountManager1.5, UserTransaction 2.3, DbConnector 1.7 and ProfileFormat 1.2, are not provided in development phase. Also, without the dependency information defined in development phase, the dependency information in release phase can not be documented properly. As the result, we can not guarantee the consistency throughout deployment phase.

Second issue is the component dependency in release phase may be reduced because developer may combine some component with a new component. For example, the dependencies of UserService 1.2 are AccountManager1.5, UserTransaction 2.3, DbConnector 1.7 and ProfileFormat 1.2 in development phase, but, in release phase, developer can combine AccountManager 1.5 and DbConnector 1.7 with UserService 1.2 so the dependencies of UserService 1.2 become only UserTransaction 2.3 and ProfileFormat 1.2. As the result, the component dependency in development phase and in deployment phase may be different.

Third issue is each organization may develop and publish the components based on different release policies. They may publish component source from repository, binary component from release server or attach component source with binary component. Developers need not only dependency information but also the method to obtain the component. For example, to develop UserService, ATech need to define the relation to AccountManager, UserTransaction and ProfileFormat. Also ATech need to define the method to get binary version of AccountManager and ProfileFormat from corresponding release sites and to get the component source of UserTransaction from repository. As the result, component developer and deployer need the method to handle with different release policies.

To summarize the problem raised in component development-deployment process, current systems come into these shortcomings:

1. Dependency information defined in development phase may not be used in release and deployment phases automatically.
2. Dependency in development phase and deployment phase may be different because of the loss of dependency information caused by combining components into a new component.
3. SCMs could not import different kinds of component developed by different organizations into local development environment automatically. The component may be packaged into the application both component source and binary format. Dependency information may not come with the components.

We need a system to manage the interrelated components in development phase and to deploy consistency set of components to middleware automatically.

Chapter 4

Our Approach

J2DEP research project addresses support in component development-deployment process. This system integrates the functionalities of configuration management, component development and component deployment together.

The key insight of this research is to manage the evaluation of third vendor component inside local configuration management. Rather than to bring and control all version of third vendor components in local configuration management, J2DEP supports developers to generate the dependency information as a dependency metadata described in chapter 5, keep and manage only versions shown by metadata in local configuration management instead.

Developers can use J2DEP to import third vendor component into development space shown in Figure 4.1 and generate dependency metadata describing about component name, version, type and method to obtain component from source repository or release site.

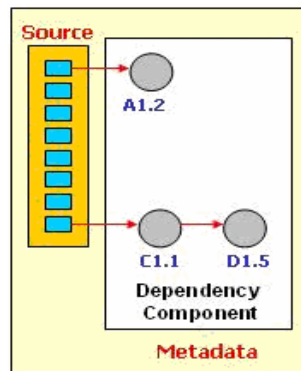


Figure 4.1: Example of Development space

Development space in J2DEP, described in Figure 4.1, is the directory structure to store source files, dependency metadata and dependency components imported by developer corresponding to dependency metadata. In development phase, developers can use J2DEP to generate dependency metadata after they import the third-vendor component into development space and they can control both source and metadata in local-SCM. The detail will be shown in implement section.

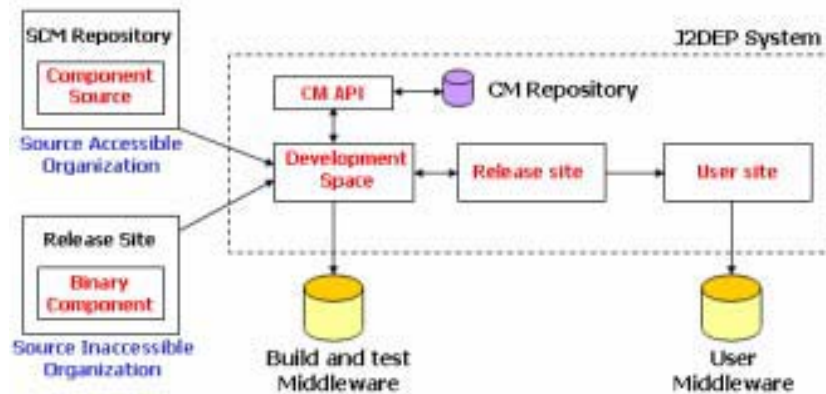


Figure 4.2: J2DEP Architecture

Figure 4.2 shows the overall of J2DEP Architecture. The development space, where the developer places the component source, dependency and perform developing. Developers can use J2DEP to import dependency component by getting source from organizations that allow to access the source from repository or by downloading the binary component from the organizations that publish only binary version. After the developer fill out the component information to J2DEP, J2DEP will generate dependency metadata into development space.

J2DEP connects development space with configuration management API (CM API, the detail is described in Chapter 6) for managing version of component sources and their dependencies and connects with release site to publish components with dependency metadata. To support consistency of component version in deployment process, J2DEP uses the dependency metadata defined in development phase.

There are 2 kinds of middleware to be deployed components.

(1) Build and test middleware is for developer to deploy the components obtained from repository and their dependencies. Developers can build a binary component and deploy it with dependencies in testing phase.

(2) End-User middleware is the middleware for end-users who deploy only binary component to operate their business requirements and do not have a relation with component development process. Deployer obtains the binary component from release site to deploy in user middleware.

J2DEP architecture consists of 2 main parts which are Development support and End-User deployment agent.

4.1 Development Support:

This functionality supports to generate dependency information for each component and connect development space with CM API, Build and test middleware and Release site

4.2 End-User Deployment Agent:

This functionality supports in deployment phase to assemble the related component to application component and deploy in end-users middleware. Also, End-User Deployment Agent will record the deployed component data to manage dependency on customer middleware.

Chapter 5

Dependency Metadata

To describe about relationship of component, developer needs to add the dependency information in development phase to development space. Dependency Metadata mainly describe about a component detail, dependency component both from source repository and release site. Developer can define the relationship among any combination of source and binary components with dependency metadata.

Dependency Metadata consists of 2 parts: (1) Metadata about component details (2) Metadata of Dependency component

5.1 Metadata about component details

This metadata describe the component detail of each component in development space. The developer has to inform the component name, vendor and type to J2DEP when he creates a new component to development space. J2DEP will generate a new metadata for a new component and assign version 1.1 automatically. Developer can change version of the component whenever he add a new CVS tag. J2DEP supports version change of component in development space.

Example

```
<!--component name -->
<componentname>UserService</componentname>
<!--component version -->
<componentversion>1.2</componentversion>
<!--vendor organization name -->
<componentvendor>ATech</componentvendor>
<!-- J2EE component type -->
<componenttype>Session Bean</componenttype>
```

Figure 5.1: Metadata about component details

Figure 5.1 shows an example of metadata for version 1.2 of UserService session bean component created by DCom company in the development space.

5.2 Metadata of Dependency component

After developer creates a component on J2DEP development space, he can add the dependency information between component in development space and component create by J2DEP system or create by the other system. The developer can import dependency component both from source repository and release site (HTTP site). To generate dependency metadata, the developer generally needs to define the details of dependency component including component name, version, vendor, type, package type and the information whether that dependency component is included in other metadata or not.

5.2.1 Metadata of Dependency component from source repository

Besides the details of dependency component, the developer needs the information about CVS source repository connection to get the component source from CVS source repository. The information consists of CVS host name, CVS root directory, CVS authentication type and CVS tag for checking out component source by tag.

Figure 5.2 shows that the component in development space is depended on the UserTransaction session bean component 2.3 created by DCom company and published in source package. To get the source package, the developer needs to connect to the location of repository which is cvshost.dcom.com in directory /work/cvsroot by using pserver authentication and tagname “UserTransaction-2.3”

5.2.2 Metadata of Dependency component from release site

To get the binary component from release site, the developer needs the information same as to get the component from source repository. Besides the details of dependency component, the developer has to inform J2DEP about the location of binary component from release site.

The dependency information in Figure 5.3 shows that the component in development space is depended on the ProfileFormat application jar component 1.2 created by CyberC company and published in binary package. To get the binary package, J2DEP will connect to HTTP server addressed www.cyberc.com/release/profileFormat.jar

Example

```
<dependencycomponent>
  <name>UserTransaction</name>
  <version>2.3</version>
  <vendor>DCom</vendor>
  <type>Session Bean</type>
  <!-- CVS server location -->
  <location>cvshost.dcom.com</location>
  <!-- dependency component package type-->
  <packagetype>source</packagetype>
  <!--this component is included metadata or not-->
  <metadata>included</metadata>
  <!-- CVS root directory-->
  <cvsroot>/work/cvsroot</cvsroot>
  <!-- CVS Authentication-->
  <authentication>pserver</authentication>
  <!-- CVS tag name-->
  <tag>UserTransaction-2.3</tag>
</dependencycomponent>
```

Figure 5.2: Metadata of Dependency component from source repository

Example

```
<dependencycomponent>
  <name>ProfileFormat</name>
  <version>1.2</version>
  <vendor>CyberC</vendor>
  <type>Application Jar</type>
  <!-- binary package location-->
  <location>www.cyberc.com/release/profileFormat.jar</location>
  <packagetype>binary</packagetype>
  <metadata>included</metadata>
</dependencycomponent>
```

Figure 5.3: Metadata of Dependency component from release site

5.3 Dependency Metadata of component from the other development systems

In case of a component developed by the other development system, it means that component does not contain the dependency metadata. So the developer has to add the dependency information from second level of dependency manually.

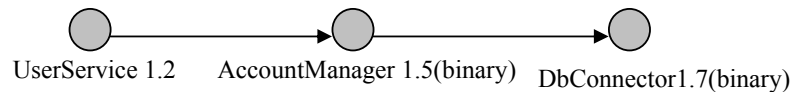


Figure 5.4: Example of dependency component from the other system

```
<dependency>
  <componentname>UserService</componentname>
  <componentversion>1.2</componentversion>
  ...
  <dependencycomponent>
    <name>AccountManager</name>
    <version>1.5</version>
    ...
    <dependon>
      <dependencycomponent>
        <name>DbConnector</name>
        <version>1.7</version>
      </dependencycomponent>
    </dependon>
    ...
  </dependencycomponent>
  <dependencycomponent>
    <name>DbConnector</name>
    <version>1.7</version>
    ...
  </dependencycomponent>
</dependency>
```

Figure 5.5: Dependency Metadata of component from the other system

Example

Developer create UseService Component likes in Figure 5.4. UserService 1.2 is depended on AccountManager 1.5 which public in binary format and AccountManager 1.5 is depended on DbConnector 1.7. However, the vendor who develops AccountManager uses the other development tool, which does not generate dependency information, so no Dependency Matadata inside the binary package. The dependency metadata is shown in Figure 5.5.

Chapter 6

Implementation

J2DEP supports variant kinds of J2EE components based on the architecture described in chapter 4. J2DEP prototype implementation is built by integrating CVS for managing source and dependency metadata, HTTP/FTP server for publishing the binary component and JBOSS middleware as a platform to deploy J2EE component.

In this chapter, we show the implementation detail of component development support and user deployment agent. First, we discuss about component development support. We show how J2DEP supports to initiate development space, import related component, generate dependency metadata, build a component, deploy component with correct dependency into middleware of development site and release component to release site. Secondly, we show how user site deployment tool support.

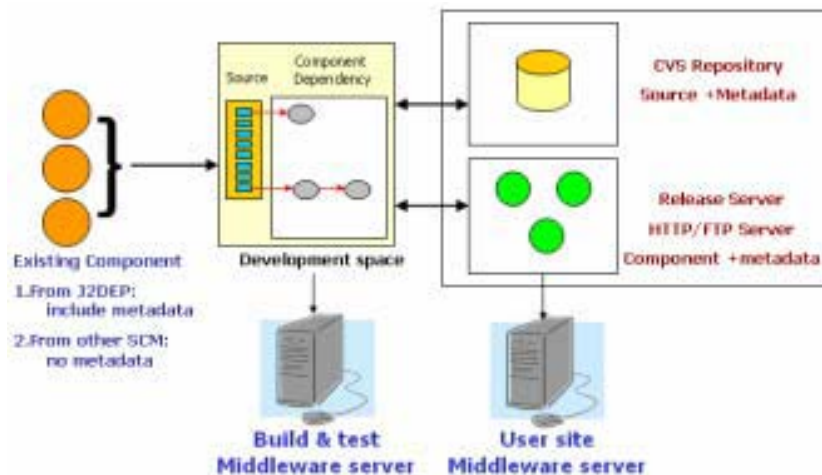


Figure 6.1: J2DEP Implementation

6.1 Development Space Structure

J2DEP build the development space to support to create component, to obtain component from source repository and to import dependency components. In the prototype systems, we use the development space like Eclipse Java development tool, so that developer can also continue developing with this tools easily. The development space structure consists of several locations which contains the following artifacts:

Source directory: component source files, dependency metadata and encrypted user name and password to connect source repository or release site of dependency components

Binary directory: compiled version of java source

Dependency component directory: binary version of dependency component imported by developer to development space

Binary version of component: the component built from corresponding sources in Source directory

Project metadata: the information about the component in development space

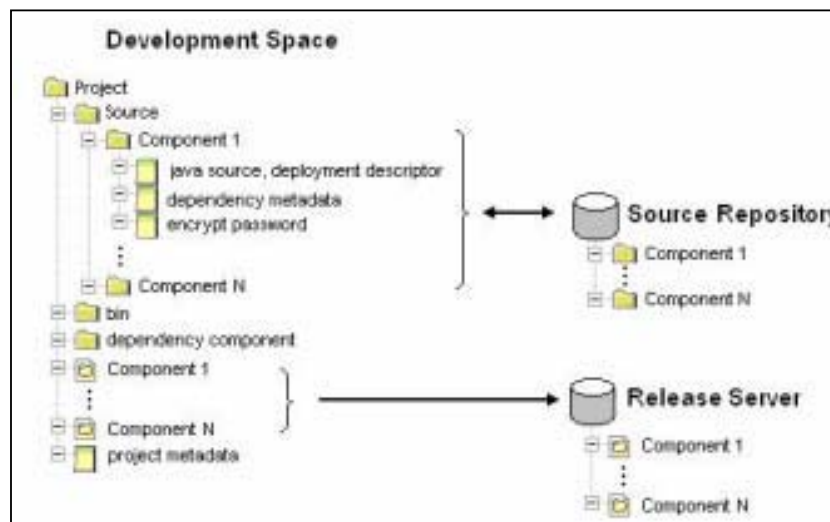


Figure 6.2: Development space, Source repository and Release site

6.2 Development Space Initialization

Developers can start to develop component by initializing a development space. J2DEP helps developer to create a new development space a new development space shown in Figure 5.2. Also, developers can open existing development space from project metadata file. Then, all components of development space defined in project metadata will be open by J2DEP tool.

J2DEP connects development space with CVS source repository shown in Figure 5.1 and 5.2. Developer can create new component or pull existing component from source repository to development space.

(1)To create new component, J2DEP supports to prepare source files and deployment descriptor for J2EE component.

(2) *To pull pre-existing component from repository*, developers have to inform J2DEP about component name and version to check out component by using CVS tagging. J2DEP supports to check out source, dependency metadata and encrypt user name and password file from source repository. Then, J2DEP will get dependency components corresponding to dependency metadata and put into Dependency component directory to prepare for build and test process in development phase.

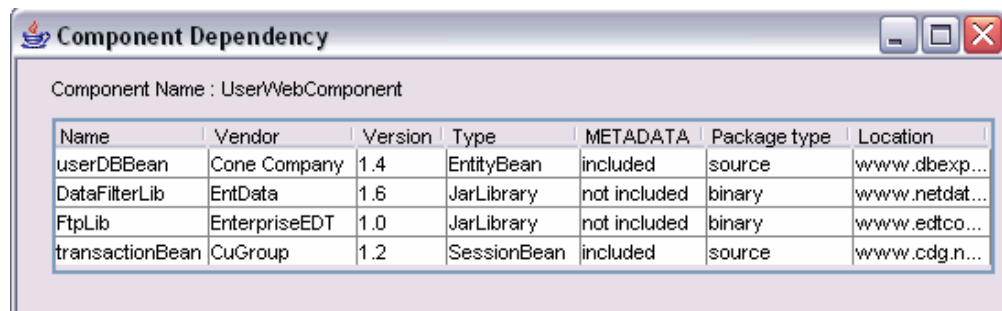
Once, developers create or check out component to development space, they can continue development by using normal configuration management procedures (e.g., using CVS)

Configuration Management API (CM API)

J2DEP supports to connect development space with configuration management. J2DEP provide the interface to import and check out component sources to SCM repository. We leave this part as the interface so that we can port J2DEP with the other kind of SCM repositories in the future. In this prototype, we currently use CVS as the SCM repository. We reuse the Apache ANT library in J2DEP to provide the CVS repository connection functions.

6.3 Dependency Component Import Tool

J2DEP help developer to import dependency component both from in-house development and third vendor component to development space. Dependency components can be component source from SCM repository or binary component from release site.

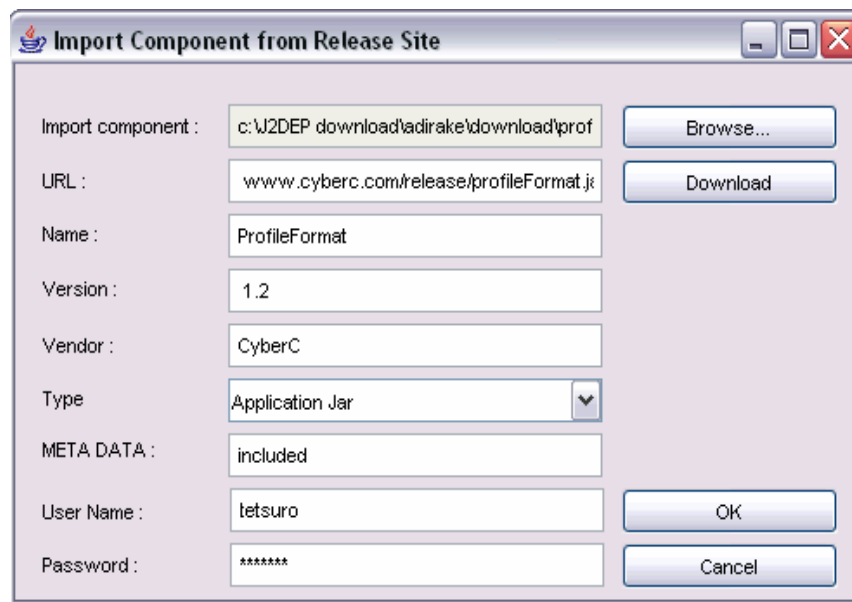


The screenshot shows a window titled 'Component Dependency' with a sub-header 'Component Name : UserWebComponent'. Below this is a table with 7 columns: Name, Vendor, Version, Type, METADATA, Package type, and Location. The table contains four rows of dependency data.

| Name | Vendor | Version | Type | METADATA | Package type | Location |
|-----------------|---------------|---------|-------------|--------------|--------------|---------------|
| userDBBean | Cone Company | 1.4 | EntityBean | included | source | www.dbexp... |
| DataFilterLib | EntData | 1.6 | JarLibrary | not included | binary | www.netdat... |
| FtpLib | EnterpriseEDT | 1.0 | JarLibrary | not included | binary | www.edtco... |
| transactionBean | CuGroup | 1.2 | SessionBean | included | source | www.cdg.n... |

Figure 6.3: Component dependencies in development space

Assume that developers want to add a new dependency to UserWebComponent in Figure 6.3, the data in the table are the dependency of UserWebComponent with their details



Import Component from Release Site

Import component : c:\J2DEP download\adira\download\prof Browse...

URL : www.cyberc.com/release/profileFormat.js Download

Name : ProfileFormat

Version : 1.2

Vendor : CyberC

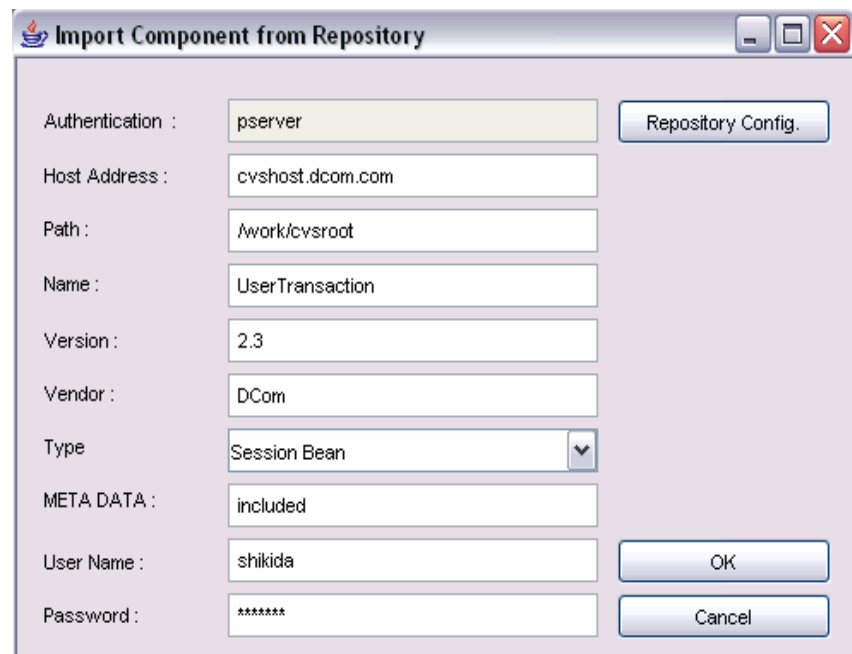
Type : Application Jar

META DATA : included

User Name : tetsuro OK

Password : ***** Cancel

Figure 6.4: Import tool for Binary Component from release site



Import Component from Repository

Authentication : pserver Repository Config.

Host Address : cvshost.dcom.com

Path : /work/cvsroot

Name : UserTransaction

Version : 2.3

Vendor : DCom

Type : Session Bean

META DATA : included

User Name : shikida OK

Password : ***** Cancel

Figure 6.5: Import tool for Component Source from repository

Then, developers have to select the method to import from repository or release site. To import dependency component, J2DEP support developer to import by 3 methods (1) download from release site (2) check out from repository (3) open from local computer. J2DEP will import component into *Dependency Component Directory*. Developer need to fill the form about component detail and connection detail shown in Figure 6.4 and 6.5.

After developer fill out the form, J2DEP will generate the dependency metadata for that imported component. For user name and password to connect repository or release server, J2DEP will encrypt them and generate a new file separated from dependency metadata.

Notice that, in case of the component does not include any metadata (e.g. the component is not built by J2DEP), developer need to add next level dependency manually.

6.4 Component Packaging and Developer Deployment Support

When developer finish developing by normal configuration management procedures, J2DEP supports to build the java source files and package compiled sources into binary component automatically.

Then, developer can use J2DEP tool to deploy or redeploy a built component and dependency components in Component Dependency Directory of development space to developer's middleware server to perform testing process. Developer can use this tool to undeploy components and their dependency from middleware.

6.5 Component Release Tool

After developer finish testing the components in development space, developer can release the binary component for developer who download components to new development space or for user who use the component in his work. To release the component, J2DEP connect with FTP server and upload the binary version of the component to the server. Developer or user can download with Development Support or User deploy agent.

In release phase, developers need to select components to combine with a component to release. If developer selects a component included in other package, the dependency of that component will be removed. For example in Figure 6.6, the dependency metadata between UserWebComponent and transactionBean are removed because transactionBean are a part of UserWebComponent.

Also, developers have to specify release policy for the component that contains sources. If they do not want to release component source, they have to inform where to get binary component to J2DEP and then Release Policy will be change to binary.

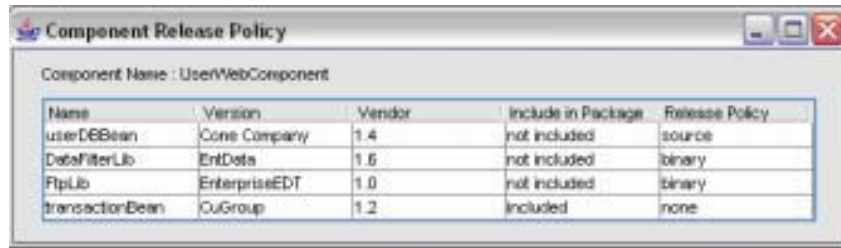


Figure 6.6: Component Release Tool

6.6 End-user Deployment Agent

End-user Deployment Agent support to deploy and undeploy the binary version of component to end-users site.

Deploy support: this function is for downloading components with their dependency component to middleware. To deploy the component, deployers have to inform the component name, version and location to download the component to J2DEP User Deployment Agent. After finishing download, User Deployment Agent will record the component that have already deployed. When deployer wants to deploy again, the existing components in middleware will not be downloaded again.

Undeploy support: this function is for removing components and their dependencies. To maintain the component consistency in middleware, J2DEP User Deployment Agent can remove only the components that is not shared by other components

Chapter 7

J2DEP Development-Deployment Process Example

7.1 Example of Development Process by J2DEP

To clarify how our system supports development-deployment process, Figure 7.1 shows a particular development by 3 organizations. ATech and DCom company develop components by using J2DEP system and DataNet company uses different development tool and SCM. Development step are following:

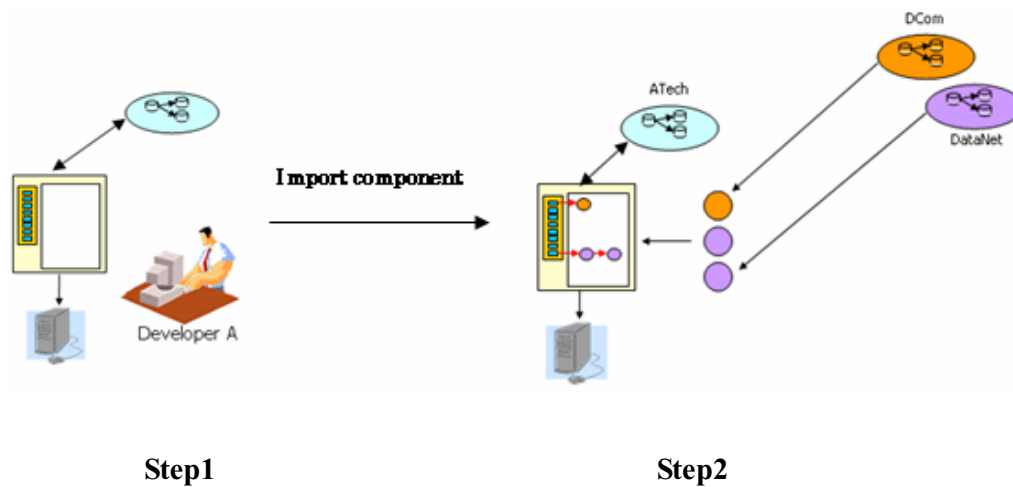


Figure 7.1: J2DEP Development Support Step 1 and 2

1. Developer A initiates development space for building component
2. Developer A imports third vendor components to local development space. To do so, developer A uses J2DEP development support tools to copy the third vendor component to his own Dependency Component directory

Third vendor component built by J2DEP system: If the component is built from J2DEP system, that component has already contained dependency metadata. So the dependency component that related this component can be also downloaded via the internet to import to

the local development space. In figure 6.1 step 2, the dependency of component from DCom will be also placed to development space. Developer needs only to add the information about where to download this component for generating dependency metadata.

Third vendor component built by other system: If the component is built by other development tool, that component has not contained any dependency metadata. So developer needs to add all dependency detail by himself. J2DEP tool supports to add the information about each component including Name, Version, Vendor, Location to download component, dependency of that component. In figure 7.1 step 2, component from DataNet is imported to development space by J2DEP tool. But developer A has to add the detail of dependency component in each level to generate metadata.

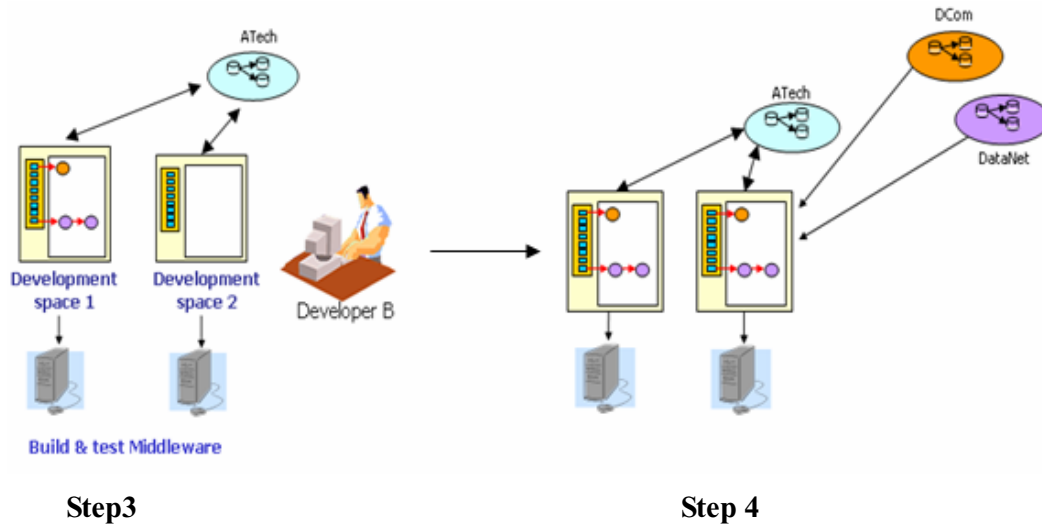


Figure 7.2: J2DEP Development Support Step 3 and 4

3. Developer B initializes new development space by check out source and dependency metadata to local development space.
4. After J2DEP tool get the dependency metadata, this tool will read the dependency metadata to get the third vendor component from the release site. Each component will be downloaded automatically from the release site and put to the development space. So developer can use that component immediately when they want to deploy the component to middleware.
5. On build and test phase, J2DEP tool supports to build component and put compiled component and dependency from development space to middleware

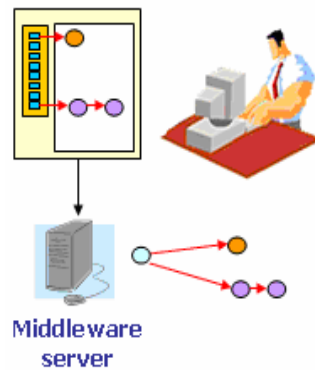


Figure 7.3: J2DEP Deployment process in developer middleware

6. After developer finishes developing and testing a component, he can release the built component to the release site. J2DEP release tool support to connect workspace to FTP server. So release manager (one of developer team) can put the binary component to the release server.

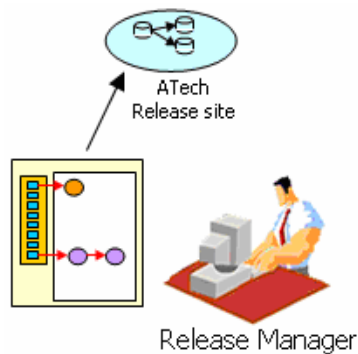


Figure 7.4: J2DEP Release process

7.2 Example Scenario of Deployment Process by J2DEP

1. Deployer informs component information to download to J2DEP Deployment agent.
2. J2DEP deployment agent downloads components and metadata from the release site. In Figure 7.5 Step 2, the component from ATech release site (HTTP server) is installed in user site's middleware. J2DEP Deployment agent will read the dependency metadata in component and prepare for downloading.



Figure 7.5: J2DEP Deployment on user middleware Step 1 and 2

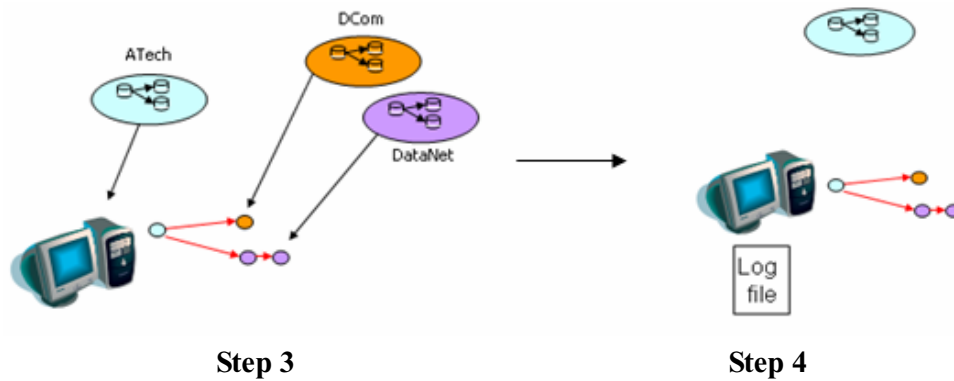


Figure 7.6: J2DEP Deployment on user middleware Step 3 and 4

3. J2DEP deployment agent automatically downloads dependency components from other release repository according to the metadata. In figure 7.6 Step 3, after deployment agent reads dependency metadata from ATech's component package, the component from DCom and DataNet will be downloaded to the middleware server automatically.
4. Record installed component to the log file to keep trace of component dependency and to be used in deploy and redeploy process.

Chapter 8

Related Works

In this chapter, we would like to discuss about related works in component based development- deployment. We show and compare the feature and the short-coming of each tool.

8.1 Eclipse[20], NetBeans[19], JBuilder[18]

These tools are generally integrated development environments (IDEs) to create simple beans, frameworks, stand-alone Java programs or a large J2EE application. They can support J2EE development by several kind of plug-ins including Component creation, Component source version control, Deployment Descriptor generation etc. Developer can use these tools for creating variant kind of J2EE components, controlling the component source in repository and testing.

Although these tools connect the development to a SCM repository, the dependency of component may not be well resolved and managed in repository. Developers do not manage the dependency information in local configuration management. Consequently, developers have to manage the dependency information by themselves whenever they check out any component sources.

Moreover, in release phase, these tools only support to release the components in source form. Developers need to create the dependency information and upload it with the binary component to the release site manually.

8.2 The RPM Package Manager (RPM)[17]

The RPM Package Manager (RPM) is a command line driven package management system capable of installation, un-installation, verification, query, and maintenance for computer software packages. RPM can support components in different versions or components developed by different organizations. The dependency information of component is described and attached in binary component by using explicit specification description file.

However, RPM do not support to manage the component in configuration management system. With RPM, we can not obtain the dependency information from configuration management repository. Also, dependency information from RPM is not included the component location. We have to obtain the dependency component manually.

8.3 Software Release Management (SRM) [15]

SRM is a tool to support software release phase. SRM helps both developers and users in the development phase and the deployment phase. Developers can release interdependent

components to their release site. Users can retrieve the component from the Internet through the SRM interface.

However, SRM supports developers to generate dependency information in the release phase rather than development phase. Also, SRM is not properly connected with configuration management system, so that, the dependency information can not be obtained from the configuration management repository.

8.4 Two-Way Integrated Configuration management and Software deployment (TWICS)[10]

TWICS is a tool which integrates Software Configuration management and Software deployment. Developer and Deployer can get a component and its dependencies from their configuration management repository. Also, TWICS supports to release a component with dependency information to the release site. Component deployer can use TWICS to deploy component and dependency together.

However, every organization has to release the component and dependency information with release tool. We can not integrate components built by an organization that did not use TWICS. We can not expect that every organization uses the same development and deployment tool.

Also, like SRM, TWICS support developers to add the dependency information in release phase rather than in development phase. This means a developer has to release the component before he decides a policy for helping the other developers.

In Figure 8.1, we compare the functionalities of each component development-deployment tools. We assume that in component development-deployment process flow consists of development phase, release phase and deployment phase.

1. Functionalities in development phase consists of: *connect the tool with development space, generate dependency, connect the tool with SCM, manage dependency information in SCM*
2. Functionalities in release phase consists of: *component release support*
3. Functionalities in deployment phase consists of: *deploy component to development space, deploy component to end-users' sites, deploy component with dependency*

In Figure 8.2, we show the functionalities of each tool that support the activities in development-deployment process flow.

| | Eclipse, Jbuilder, Netbeans IDE | RPM | SRM | TWICS | J2DEP |
|---|--|------------|------------|--------------|--------------|
| 1.1 Connect the tool with development space | O | X | X | Δ_3 | Δ_4 |
| 1.2 Generate dependency information | X | Δ_1 | Δ_2 | Δ_2 | O |
| 1.3 Connect the tool with SCM | O | X | X | O | O |
| 1.4 Manage dependency information in SCM | X | X | X | O | O |
| 2.1 Component release support | X | O | O | O | O |
| 3.1 Deploy component to development space | O | O | O | O | O |
| 3.2 Deploy component to end-users' sites | X | O | O | O | O |
| 3.3 Deploy component with dependency | X | X | O | Δ_3 | O |

O = supported, X = not supported, Δ = partially supported (please see the remark)

*** Remark

Δ_1 Generate dependency but no detail about component location

Δ_2 Generate dependency information only in release phase

Δ_3 Support only component built by TWICS

Δ_4 Target space must be same as Eclipse development space in development phase

Figure 8.1: Functionalities of J2DEP and related tools

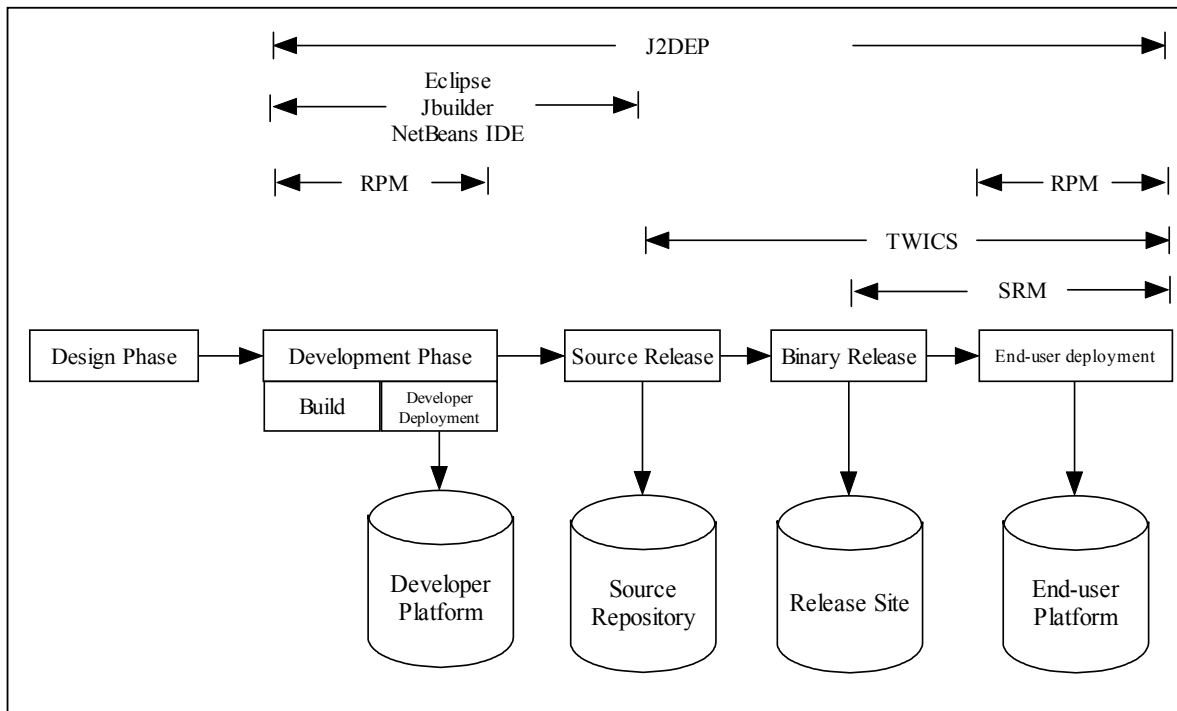


Figure 8.2: Comparing each tool on Development-Deployment process flow

Chapter 9

Conclusion

In this research, we proposed J2DEP, the integration of component-based development-deployment and software configuration management in middleware. This system is designed to support managing the interrelation of in-house components and third vendor components. J2DEP links between the component development and component deployment. The dependency metadata defined in development phase and component source can be controlled in local configuration management. J2DEP uses dependency metadata to support component building, releasing process of developers. Also, J2DEP helps developer or deployer to deploy consistency set of components in both development site and user site.

Further work

We build J2DEP as a prototype system to support J2EE development-deployment automatically. However, the key aspect to manage dependencies of interrelated components is still not completed. For example, developers have to merge different versions of dependency metadata manually. J2DEP is currently being developed. In the future, we intend to extend J2DEP functionalities to support the shortcoming.

In current prototype, J2DEP is a tool separated from the other development tools. This would be tedious for developer to use. In the future, J2DEP should be implemented as an Eclipse plug-in. This would allow developers to gain the benefit from all infra-structure provided by Eclipse.

Acknowledgments

The research can be success by good advices and supports. I am really appreciate Professor Koichiro Ochimizu and Professor Yoichi Shinoda for advices about J2DEP research, Associate Professor Masato Suzuki for the idea and concept of component-based development, Associate Kazuhiro Fujieda supported me knowledge about software configuration management and component-based development-deployment and Associate Satoshi Hattori supported me the facilities in laboratory.

Also, I would like to thank Ryo Hayasaka for helping me to translate Japanese abstract and Saw Sanda Aye for a good comment on my research and aspiration to do J2DEP research.

References

1. B.J. Rumbaugh, Developing Enterprise Java Applications with J2EE and UML, Addison-Wesley, 2002
2. I. Singh, B. Stearns, M.J. Son, Design Enterprise Applications with the J2EE Platform, SUN Microsystems, 2002
3. Microsoft .NET, <http://www.microsoft.com/net/technical/>
4. Common Object Request Broker Architecture (CORBA), <http://www.corba.org/>
5. K. Whitehead, Component-based Development, Addison-Wesley, 2002
6. H. Cervantes and R.S. Hall, Autonomous Adaptation to Dynamic Availability Using a Service-Oriented Component Model
7. C.Szyperski, Component Software, Addison-Wesley, 1998
8. D.W. Weber, Requirements for an SCM Architecture to enable Component-Based Development, Proceedings of the 10th International Workshop on SCM, 2001
9. S.H. Edwards Software Component Relationships, the 8th Annual Workshop on Institutionalizing Software Reuse, 1997
10. S. Sowrirajan and A. van der Hoek, Managing the Evolution of Distributed and Interrelated Components, Proceedings of the 11th International Workshop on SCM, 2003.
11. M. Larsson, I. Crnkovic, Configuration Management for Component-based Systems, Proceedings of the 10th International Workshop on SCM, 2001
12. RCS, <http://www.gnu.org/software/rcs/rcs.html>
13. CVS, <https://www.cvshome.org/>
14. R.S. Hall, D. Heimbigner, A.L. Wolf, A Cooperative Approach to Support Software Deployment Using the Software Dock, Proceedings of the 1999 International conference on Software Engineering, 1999
15. A. van der Hoek, R.S. Hall, D. Heimbigner, and A.L. Wolf Software Release Management,
16. D.C. Schmidt and S. Vinoski, The CORBA Component Model: Part 1, Evolving Towards Component Middleware, *C/C++ Users Journal*, February 2004
17. RPM, <http://www.rpm.org>
18. JBuilder, <http://www.borland.com/jbuilder/>
19. Netbeans IDE, <http://www.netbeans.org/>
20. Eclipse, <http://www.eclipse.org>
21. Apache ANT, <http://ant.apache.org/>

Appendix

dependency.dtd

```
<!DOCTYPE component-dependency PUBLIC
    "-//Adirake Pimruang//DTD J2EE//EN"
    "http://www.jaist.ac.jp/~p-adirake/J2DEP/dependency.dtd">
<!ELEMENT dependency (componentname, componentversion, componentvendor,
    componenttype, component*)>
<!ELEMENT componnetname (#PCDATA)>
<!ELEMENT componentversion (#PCDATA)>
<!ELEMENT componentvendor (#PCDATA)>
<!ELEMENT componnettype (#PCDATA)>
<!ELEMENT dependencycomponent (name, version, vendor, type, location, packagetype,
    metadata, cvsroot, authentication, tag, dependon)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT version (#PCDATA)>
<!ELEMENT vendor (#PCDATA)>
<!ELEMENT type (#PCDATA)>
<!ELEMENT location (#PCDATA)>
<!ELEMENT packagetype (#PCDATA)>
<!ELEMENT metadata (#PCDATA)>
<!ELEMENT cvsroot (#PCDATA)>
<!ELEMENT authentication (#PCDATA)>
<!ELEMENT tag (#PCDATA)>
<!ELEMENT dependon (dependencycomponent*)>
```
