Journal of Software Engineering

ISSN 1819-4311

Academic Journals Inc.

www.academicjournals.com

Journal of Software Engineering 2 (1): 23-38, 2008
ISSN 1819-4311

A Proposition of Generic Deployment Platform for Component Based Applications

[1]A. Benamar, [2]N. Belkhatir and [1]F.T. Bendimerad
[1]Universite Aboubekr Belkaid de Tlemcen, BP 230, Tlemcen 13000, Algérie
[2]Adele Team, LSR-IMAG 220 Rue de la Chimie, Grenoble 38041, France

Abstract: In this study, we propose a generic approach for the deployment of component based applications. The proposed approach is based on D and C specification and model driven architecture that is becoming a key approach for model transformation. Specifically, our generic deployment tool is implemented and tested for enterprise Java beans through two main steps. Starting from a platform specific model, it is transformed by means of Atlas transformation language to a platform independent model that is represented by D and C application model. Then, the resulting D and C model and deployment domain model allow to launch the deployment plan by describing the choice of components, their locations and linkages.

Key words: Deployment, enterprise java beans, D and C specification, model driven architecture

INTRODUCTION

Software life cycle is composed of many steps such as analysis, design, production, test, deployment and execution. With the emergence of the component oriented programming paradigm, each life-cycle step is subject to abstraction for reuse and standardization. This study subject focuses on the deployment step in the context of component based applications. Complexity and importance of deployment are both increased by network evolution and component based applications. Many component technologies such as the Object Management Group (OMG) CORBA Component Model (CCM), the Sun Microsystems Enterprise Java Beans (EJB) and the Microsoft corporation. Net exist. But generally, the proposed systems are difficult to modify and hard to configure. Also, even if the deployment logic and its concepts are almost always the same ones, one can easily note that each component technology proposes its own deployment system. Therefore, we would like to propose a generic approach for a deployment and reconfiguration system. A generic approach makes it possible to concentrate on deployment concepts and to implement a common and reusable platform.

In this study, we propose a generic approach that allows to provide a deployment platform for component based applications. This approach follows the Model Driven Architecture (MDA) technique and identifies two main steps:

- Transformation of Platform Specific Model (PSM) to Platform Independent Model (PIM) by means of Atlas Transformation Language (ATL)
- Execution of deployment plan, by affecting and connecting the component systems to the network

RELATED WORK

There is a considerable interest in development of generic deployment model that is relevant to MDA approach and metadata management.

Corresponding Author: A. Benamar, Universite Aboubekr Belkaid de Tlemcen, BP 230, Tlemcen 13000, Algérie

Lestideau and Belkhatir (2003) tried to solve the deployment process of the component-based applications. They defines a component model that is used for deployment purposes and that should fit for the most component models. But this model is not sufficient; it does not cover all the necessary abstractions typical for software components. The authors use the model for configuration purposes, i.e., to select the concrete components that form an application, where their model is sufficient. But for the purpose of the whole deployment process, the model is insufficient (for example it does not cover ties among components).

Quema *et al*. (2004) designed an environment for deploying software components. They define own component model and own ADL for manipulating components. The focus is mainly imposed on scalability and fault-tolerance of the environment.

Kebbal and Bernard (2001) also defined an environment for deploying components. The authors focus mainly on discovery of components (based on trading). The solution is integrated into CCM.

Merle and Belkhatir (2004) proposed an Open enviRonment to deploY Application (ORYA) which provides distributed support for deploying (ordinary) application. ORYA supports install, configure, reconfigure and de-install stages of deployment process. Because it supports deployment of ordinary applications, the planning stage is very primitive (for ordinary applications, there are usually no necessities for complex planning).

Deng *et al*. (2005) proposed an implementation of D and C specification called Deployment And Configuration Engine (DAnCE). The engine is currently under development and does not support all features from the specification. It allows deployment of CCM components.

Hnětynka (2004) presented a Deployment Factory (DF) and model-driven unified environment for deploying distributed component based applications. The DF via its generic feature support heterogeneous applications is based on D and C specification. Moreover, DF is built using a plug-in concept and facilitates deployment of components of most of the contemporary component technologies without the necessity to modify them.

Flissi and Merle (2004) proposed a multi personalities environment for configuration and deployment of component based applications. This environment is composed of a core capturing a canonical model of configuration and deployment and a set of personalities customized with languages and platforms. Moreover, the proposed system details the architecture of such environment and describes the personalities for the CORBA and Fractal component models.

SOFTWARE DEPLOYMENT PROCESS

According to Carzaniga *et al*. (1998), the term of software deployment refer to all the activities that make a software system available for use. These activities represent a life cycle of an application and are related to the release, install, activate, de-activate, update, adapt, de-install and de-release. Although we can identify a set of distinct and key activities that typically constitute a generic deployment process, we cannot precisely define the particular practices and procedures embodied in each activity. They heavily depend on the characteristics and requirements of the software producers and consumers. Therefore, Fig. 1 should be interpreted as a reasonable process that has to be customized according to specific requirements of the deployment activity being observed.

- **Release:** This is the most important activity for software deployment process. It represents an interface between deployment and development process. Thus, it must determine the resource required by the software system to operate correctly and must also collect the necessary information for carrying out subsequent activities
- **Install:** The installation activity covers the initial insertion of a system into a consumer site. Effectively, it is the most complex of the deployment activities because it deals with the proper assembly of all the resources needed to use a system. It is also currently the activity best supported by specialized tools

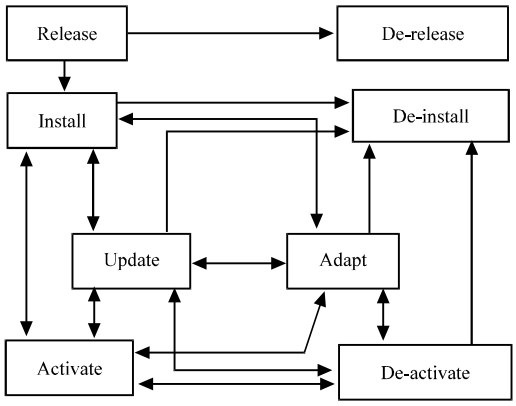

Fig. 1: The software deployment process (Carzaniga *et al*., 1998)

- **Activate and de-activate:** these activities consist respectively to start up and shutdown the executable system. For a simple system, activation involves establishing some form of command (or clickable graphical icon) for executing the binary software. For a complex system, it might be necessary to start servers or daemons, before the software system itself is activated. Furthermore, de-activation is often required before other deployment activities
- **Update:** It is a special case of the install activity. However, it is usually less complex because it can often rely on the fact that many of the needed resources have already been obtained during the installation process. Similar to installation, update also includes the transfer and configuration of the components needed to complete the operation
- **Adapt:** Like the update activity, the adaptation involves to modify a software already installed. Adaptation differs from update in that the update activity is initiated by remote events, such as a software producer releasing an update, whereas adaptations are initiated by local events, such as a change in the environment of the consumer site. Therefore, the same release is kept but only modified in terms of configuration
- **De-install:** It consists of removing an installed application. Of course, de-installation presumes the system is also deactivated. This activity must be done properly, because it is possible that, it involves some reconfiguration of other systems in addition to the removal of the de-installed system's files
- **De-release:** Finally a system is marked as obsolete and support by the producer is withdrawn. This requires that the withdrawal be advertised to all known consumers of the system

SURVEY OF DEPLOYMENT PLATFORMS FOR COMPONENT BASED APPLICATIONS

Every component technology uses its own component model. This section provides an overview and analysis of the typical component models used in contemporary component technologies (Wang and Qian, 2005). Specifically, we have considered the OMG's CCM, the Sun Microsystems EJB and the Microsoft Corporation. Net.

Corba Component Model (CCM)

CCM is a component specification developed by OMG. The main focus of CCM is to ease the development process of applications made of distributed components that are heterogeneous. In the initial definition of Corba architecture, the aspects related to the deployment were ignored. The specification was exclusively oriented towards the object interoperability.

However, in the new component specification, the OMG has given a great importance to the software deployment and distribution. In fact, the specification includes a deployment model allowing to describe component assemblies. Four models constitute the Corba 3.0 standard, including abstract model, programming model, deployment model and execution model.

- **Abstract model:** It covers specification of component interfaces and their interconnections
- **Programming model:** It describes component implementations and their non-functional properties
- **Deployment model:** It defines how to assemble component systems (i.e., packaging, installing)
- **Execution model:** It defines containers as a runtime environment for component instances

Enterprise Java Bean (EJB)

The EJB framework, provided by Sun Microsystems, is a component architecture intended for the development of distributed, object-oriented business applications in the Java programming language. EJB component (Monson-Haefel and Burke, 2006), also called bean is a server component represented by home interface and the remote interface. The remote interface reflects the functionality of the bean and is implemented by so named business methods. On the other hand, the home interface supports methods for creation and removal of particular bean objects, as well as methods for querying the population of the EJB objects. A container provides a deployment environment that wraps the beans during their life cycle; every bean lives within a container. The container provides services that the contained beans can use namely transactions, security and persistence. By default, both interfaces are accessible via Remote Method Invocation (RMI). Every client method invocation on a bean is supervised by the bean's container, which makes it possible to manage the transactions according to the transaction attributes that are specified in the corresponding bean's deployment descriptor.

Microsoft's .Net

The newest significant product in the area of distributed computing is the Microsoft's .Net Framework. Although its concepts are far different from the traditional concepts of the component-oriented development, it can provide quite a flexible way to develop distributed applications. Basic building units in the .NET Framework are classes, sometimes also called components. The source code of the classes can be implemented in variety of programming languages, including C#, visual Basic .Net and Jscript. Once finished, it is compiled into the Microsoft Intermediate Language (MSIL). The structure of the resulting files is very similar to the structure Java class files. As a next product of the compilation process, a manifest file with description of published data types, dependencies and digital signatures is created. All the resulting files are packaged into a single file called an assembly. The assembly files must be manually deployed to the target machine and run with the support of the Common Language Runtime (CLR).

Synthesis

As shown in the subsequent tables, this synthesis is focused on two aspects. The first aspect is related to metadata (Table 1) and the second aspect is related to the activities provided by platform interfaces (Table 2).

Deployment Metadata

- **Component Implementations:** All the deployment technologies studied above are based on the package concept. A package is an archive that contains the component files together with some metadata describing the component systems. For instance, CCM allows to use multiple

Table 1: Comparison of deployment platforms through deployment metadata

	CCM	EJB	.Net
Component implementations	Support multiple implementations. For each, description, reference to its code, compiler name, dependences, programming language, location constraints	Java classes	List of implementation files (C#, Visual Basic .Net or Jscript)
Component structures	Abstract models of component (facets, receptacles, events and attributes) and its home	Provided interfaces and home interfaces	No structural information in manifest
Component dependences	Packages, resources and files	Resources	Static references to assemblies and resources
Component functional properties	Description of configuration properties	Description of configuration properties	Not managed
Component non-functional	Type of component, transaction, persistence, QoS of event ports and threading.properties	Type of component, transaction, persistence, relations and attributes provided by container	Not managed
Deployment domain modeling	Not supported	Not supported	Not supported
Deployment plan	Supported	Not supported	Not supported
Abstract assembly description	Not supported	Supported	Not supported

Table 2: Comparison of deployment platforms through deployment activities

	CCM	EJB	.Net
Package installation	Manual	Manual	Manual
Instantiation	Supported	Supported	Supported
Connection	Supported	Supported	Supported
Site assignation	Not supported	Not supported	Not supported
Assembly supervision	Not supported	Not supported	Not supported
Activation/De-activation	Supported	Supported	Supported
De-installation	Manual	Manual	Manual

implementations of the same components. Besides, the selection of suitable implementation is carried out at runtime. However EJB and .Net are restrained to a single component implementation. In fact, EJB and .Net depend on specific technologies that are respectively Java Virtual Machine (JVM) and CLR

- **Component structures:** Almost the deployment platforms allow to describe the structural information of the component excluding .Net. Moreover, some conception and assembly tools would highlight component interfaces and so that detect the possible connection among these connections
- **Component dependences:** All the deployment platforms let to represent component dependences among packages and resources. For instance, CCM could support a general description of component dependences, besides the dependences through the same component implementation
- **Component properties:** Thanks to non-functional properties, CCM and EJB could define container type, so that benefiting of functional services (transaction, persistence and security). However, the .Net deployment model is unable to characterize the non-functional properties and its manifest file could not describe the functional properties
- **Domain modeling:** All the deployment technologies studied above have no feature to model the deployment domain
- **Deployment plan and assembly description:** Even if, CCM support a suitable description of deployment plan, EJB support only the assembly description. On the other side, the concept of assembly in .Net is only based on possessing a reference to its implementation

Deployment Activities

All the deployment platforms studied previously, realize the basis deployment activities such as instantiation, configuration and connection, but more or less expressed differently. For instance in CCM, the deployment activities are opaque, which in turn decreases the dynamicity and flexibility of deployment process.

GENERIC DEPLOYMENT PLATFORM

The deployment of component based applications is not unified. Moreover, each component model solves the deployment issues in its own proprietary way. Therefore, we propose a generic approach that is intended to make unified the deployment of component based applications. Specifically, this approach is based on model transformation technique that uses an appropriate PIMs and PSMs.

Model Transformation Overview

Several attempts were performed for making the deployment of software component unified but none of them supports all the necessary features. OMG tries to solve this problem in its Deployment and Configuration (D and C) of component based distributed applications specification (OMG, 2004). The specification follows the MDA paradigm. It defines an approach to software development based on modeling and automated mapping of models to implementations. The basic MDA pattern involves defining a PIM and its automated mapping to one or more PSMs. While the current OMG standards such as the Meta Object Facility (MOF) and the Unified Modeling Language (UML) provide a well-established foundation for defining PIMs and PSMs, no such well established foundation exists for transforming PIMs into PSMs (Gerber *et al*., 2002). In 2002, in its effort to change this situation, the OMG initiated a standardization process by issuing a Request for Proposal (RFP) on Query/Views/Transformations (QVT). This process will eventually lead to an OMG standard for defining model transformations, which will be of interest not only for PIM-to-PSM transformations, but also for defining views on models and synchronization between models.

Driven by practical needs and the OMG's request, a large number of approaches to model transformation have recently been proposed. In the following, we present a number of existing model transformation approaches: VIATRA (Varró *et al*., 2002), Tefkat (Gerber *et al*., 2002), AMW (Bézivin *et al*., 2005), ATL (Jouault and Kurtev, 2005; Bézivin and Jouault, 2006), Kent (Akehurst and Kent, 2002) and C-SAW (Gray *et al*., 2006).

Platform Modeling

This subsection explains the overall design and functionality of generic deployment platform for component based applications. For this purpose, present system uses OMG's D and C models and EJB PSM that are presented hereunder.

OMG's D and C Specification

OMG's D and C specification is an attempt to define generic support for deployment of component-based software. This specification defines three main PIMs (e.g., component model, target model, execution model). Moreover, each model has two parts, or it can be said views on the model (e.g., data, management).

Besides component packages and configuration, the data part of the component model defines descriptors for components with their interfaces and implementations. Figure 2 provides a simplified view of the component model.

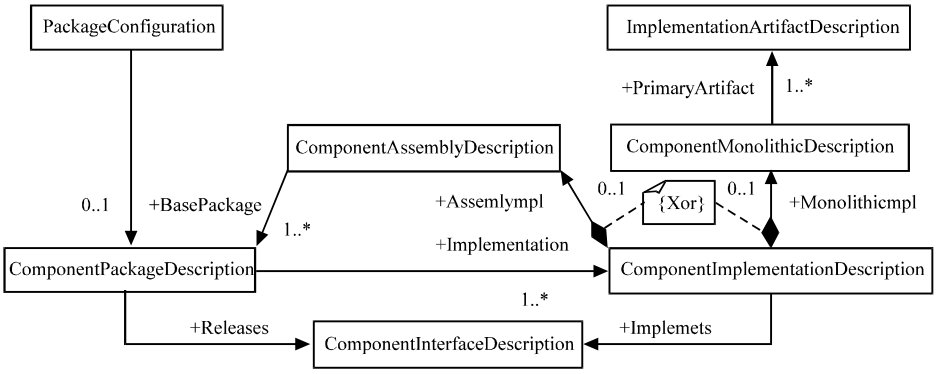

Fig. 2: OMG's D and C component data model (OMG, 2004)

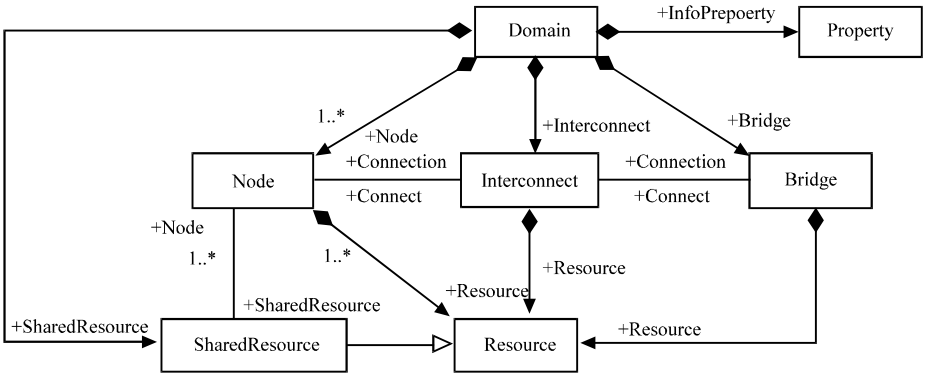

Fig. 3: OMG's D and C target data model (OMG, 2004)

Package Configuration describes a configuration of a component (i.e., the result of configuration stage). ComponentPackageDescription describes the component packed in a package. It refers to ComponentInterfaceDescription, which describes provided and required interfaces of the component and to ComponentImplementationDescription, which can be either monolithic implementation (ComponentMonolithicImplemtation) or assembled from other components (Component AssemblyImplemtation). A monolithic implementation is composed from implementation artifacts (code of the component, etc.), whereas an assembly implementation is composed only from other interconnected components; it does not directly contain any implementation artifact.

The management part of the component model contains just single class called Repository. It stores and manages information about components. It provides API for installing component packages into the repository, searching the repository content by different criteria and also for deleting packages.

Figure 3 shows the data part of the Target model, which describes and manages information about the target site into which applications can be deployed.

In the specification, the target site is called Domain. A Domain comprises of Nodes, Interconnects, Bridges and SharedResources. Nodes provides a computational capability and instances of components are executed on them. Also, nodes have resources and shared resources (shared among several nodes) and requirements of components are satisfied with these resources. Interconnects directly connect two nodes and do not have only shared resources. Bridges route component connections between interconnections. A minimal domain consists of just a single node.

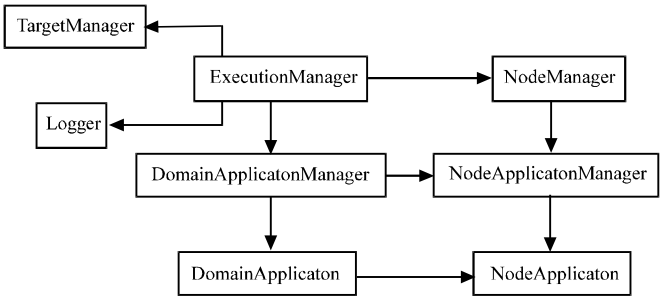

Fig. 4: OMG's D and C execution management model (OMG, 2004)

The management part of the Target model contains just TargetManager that provides information about the domain using data model and allows querying, allocating and releasing resources within the domain.

The data part of the Execution model defines the DeploymentPlan, which is produced by planning stage and describes actual deployment decisions about concrete application. The DeploymentPlan is very similar to ComponentPackageDescription from component model; it can be viewed as a flattened component assembly. Each composite component is recursively replaced with their subcomponents and finally, the deployment plan consists just of primitive components, respectively implementation artifacts and their connections. All artifacts and connections have assigned nodes and interconnections from target model.

After the planning stage, the application is ready to be launched according to deployment plan via ExecutionManager from the management part of the Execution model. An overview of this model is shown on Fig. 4.

There is only one ExecutionManager for the whole domain. It manages execution of applications and has knowledge of NodeManages, which manage nodes and also manage a part of the application executed on this node. Each launched application has a DomainApplicationManager that controls the launching process of the application in the scope of the domain and also it is used to stop application. On the other hand, NodeApplicationManager controls the launching process on the corresponding node. Finally, the whole executed application is represented by DomainApplication; NodeApplication represents piece of the application that is executed within a single node.

EJB PSM

The current version of our system supports only EJB PSM, because this model is commonly used by academic and industrial communities and provides almost features related to component based applications. A graphical overview of EJB PSM Metamodel is shown in Fig. 5.

Implementation

Present prototype is implemented with Eclipse SDK (Version: 3.2.1, Build id: M20060921-0945). Specifically, the prototype uses respectively application and domain metamodels of D and C specification as PIM and domain metamodels. Actually, only EJB PSM is tested and a graphical overview of its transformation process is shown in Fig. 6.

The main tools used for prototype implementation are presented in the following:

- Eclipse Modeling Framework (EMF) as development environment
- Atlas Transformation Language (ATL) as EMF plug-in for model transformation
- Eclipse Web Tools Platform (WTP) as Eclipse plug-in for EJB implementation
- Ant Build Tool (ABT) as running tool

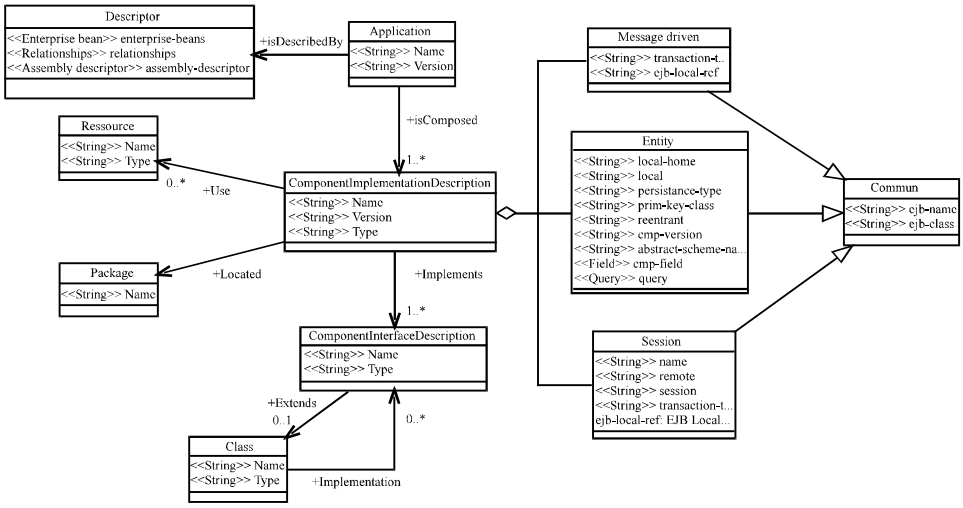

Fig. 5: Metamodel of EJB PSM

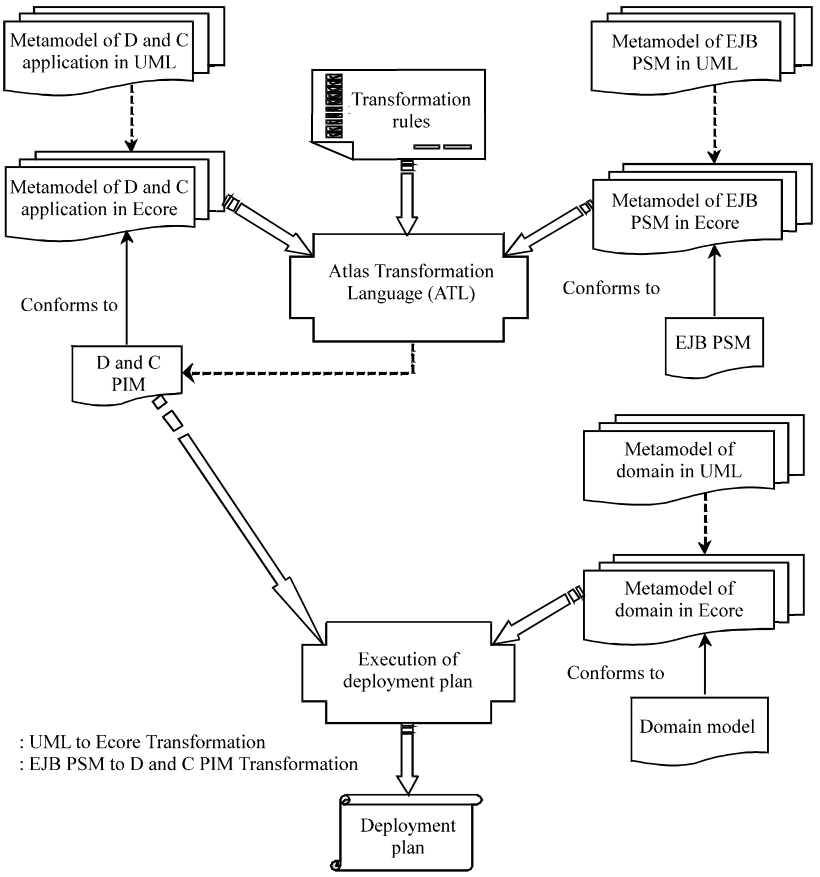

Fig. 6: Stages of metamodel transformations

Eclipse Modeling Framework (EMF)

Eclipse was created by OTI and IBM teams responsible of Integrated Development Tool (IDE) products. Eclipse is platform that has been designed for building integrated web and application development tooling. At the core of Eclipse there is an architecture for dynamic discovery, loading and running of plug-ins. Each plug-in can then focuses on a small number of well-defined tasks.

EMF is a modeling framework and code generation facility for building tools and other applications based on a structured model. EMF provides its own format (Ecore) for models and metamodels encoding. This format is based on the semantics of the Ecore metamodel and the corresponding files are encoded with XMI 2.0. Although possible, the manual edition of Ecore metamodels is particularly difficult with EMF. In order to make this common kind of editions easier, the ATL Development Tools (ADT) include a simple textual notation dedicated to metamodel edition, called Kernel MetaMetaModel (KM3). This textual notation greatly eases the edition of metamodels. Once edited, KM3 metamodels can be injected into the Ecore format using ADT integrated injectors.

In this case, the prototype is developed via Eclipse project, which include the following file formats:


KM3: It allows to introduce metamodels through text editor


Ecore: It provided by EMF injector KM3 to Ecore tool


ATL: It is composed of rules that define how EJB PSM is matched to create D and C PIM


XML: It describes deployment metadata and is represented by deployment descriptor of EJB PSM

Figure 7 and 8 provide, respectively a project explorer view and a graphical overview of EJB PSM metamodel in Ecore format.

Recall that application and domain metamodels are provided by D and C specification; also their graphical overviews in Ecore format are shown in Fig. 9 and 10.

Atlas Transformation Language (ATL)

ATL is the LINA-INRIA research group's answer to the OMG MOF/QVT RFP. It is a model transformation language specified as both a metamodel and a textual concrete syntax. Developed over the Eclipse platform, the ATL IDE provides a number of standard development tools (syntax highlighting, debugger, etc.) that aim to ease the design of ATL transformations. The ATL development environment also offers a number of additional facilities dedicated to models and metamodels handling. These features include a simple textual notation dedicated to the specification of metamodels, but also a number of standard bridges between common textual syntaxes and their corresponding model representations.

In this case, the aim of using ATL is to design a set of rules that define how EJB PSM is matched to create D and C PIM. The Figure 11 provides a screenshot for the corresponding ATL file.

Web Tools Platform (WTP)

The Eclipse Web Tools Platform (WTP) project, seeded by contributions from the IBM Rational Application Developer for WebSphere and ObjectWeb Lomboz, provides a set of well-rounded and tightly integrated tools that simplify the creation of often complex web and J2EE applications. The WTP consists of two subprojects: Web Standard Tools (WST) and J2EE Standard Tools (JST). The WST project contains tools for programming-language-neutral standards such as HTML, XML and web services. The JST project contains tools specific to the Java language and its J2EE platform such as EJB, Servlet and Java Server Pages (JSP) and Java web services.

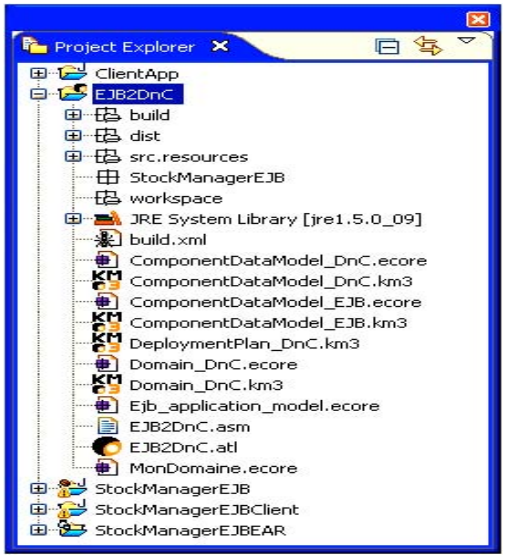

Fig. 7: Project explorer view

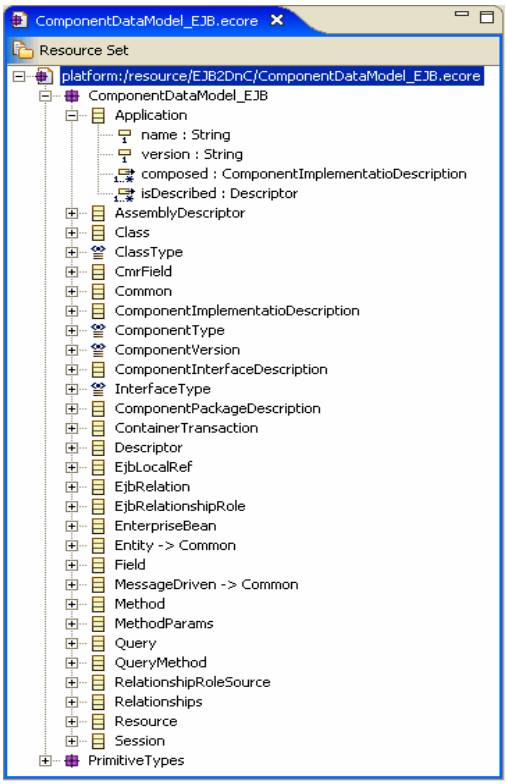

Fig. 8: EJB PSM metamodel (Ecore)

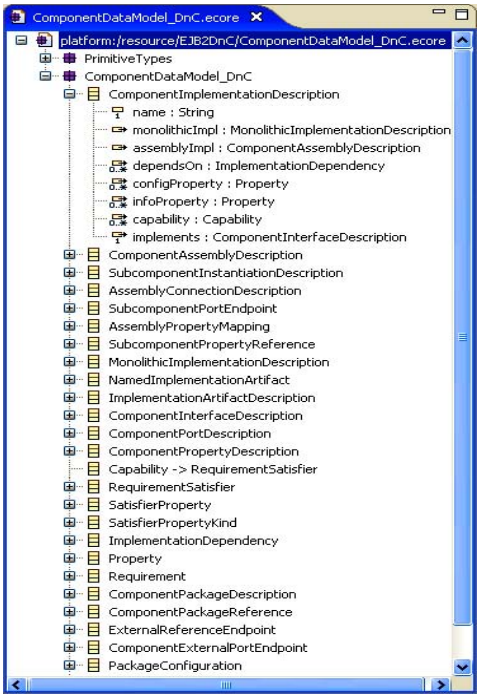

Fig. 9: D and C application metamodel (Ecore)

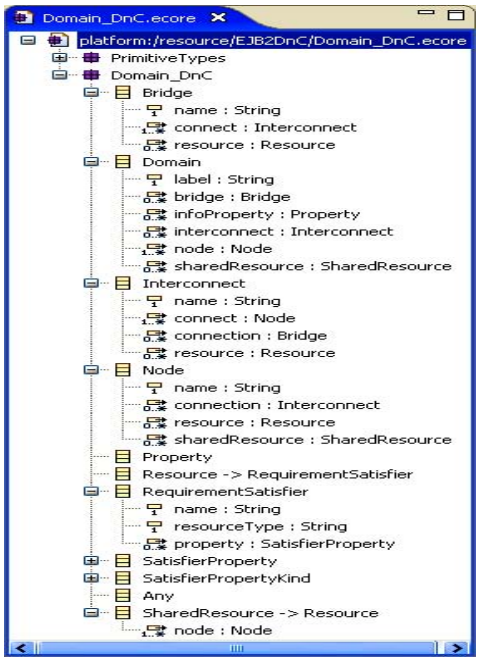

Fig. 10: D and C domain metamodel (Ecore)

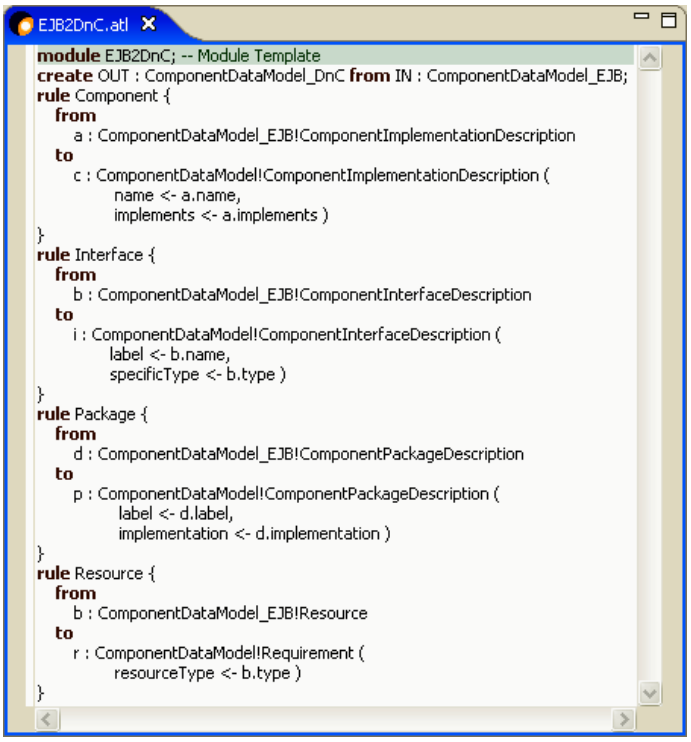

Fig. 11: ATL file EJB2DnC.atl

Recall that our prototype aims to deploy EJB specific application. For this reason, we use WTP to develop an EJB application of stock management, which consists of three projects:

StockManagerEJB (application project) is used to organize the resources (e.g., XML deployment descriptor, Manifest. MF file) and contains EJB module (i.e., StockManager.ejb). This EJB module is installed in enterprise bean container and includes entity beans (e.g, City, Country, Product, ProductFamilly, Provider) and session bean (e.g., StorageServiceBean).

StockManagerEJBClient (client project) contains all the interface classes that a client program needs to use the client views of the enterprise beans.

StockManagerEJBEAR (Enterprise Archive EAR project) includes all deployment metadata of client and application project which are archived in deployment descriptor (application.xml).

A graphical overview of EJB specific application is shown in Fig. 12.

Ant Build Tool (ABT)

ABT is Eclipse's external tool framework and is developed by Apache software foundation. ABT is a Java-based build tool that uses build files written in XML. The build files use a target tree where various tasks are executed. A target, which is a set of tasks to be executed, can depend on other targets. Examples of targets include creating directories, compiling, packaging into Java ARchive (JAR) for distribution, deploying and so on.

In this case, once ATL transformation tool generates D andC PIM, then the next step is to prepare and execute the deployment plan. For this purpose, we use the Ant build file Build.xml. Mainly, this file executes the following tasks:

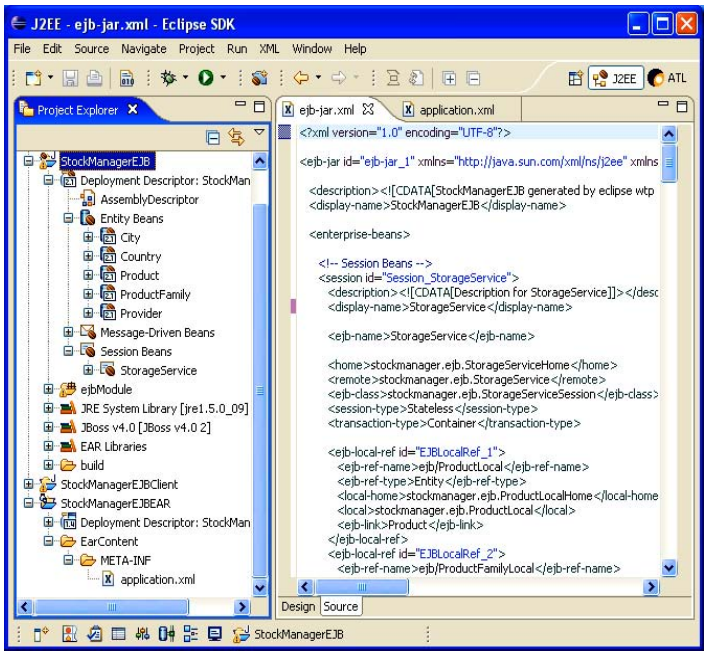

Fig. 12: Project explorer view of EJB specific application (StockManagerEJB)

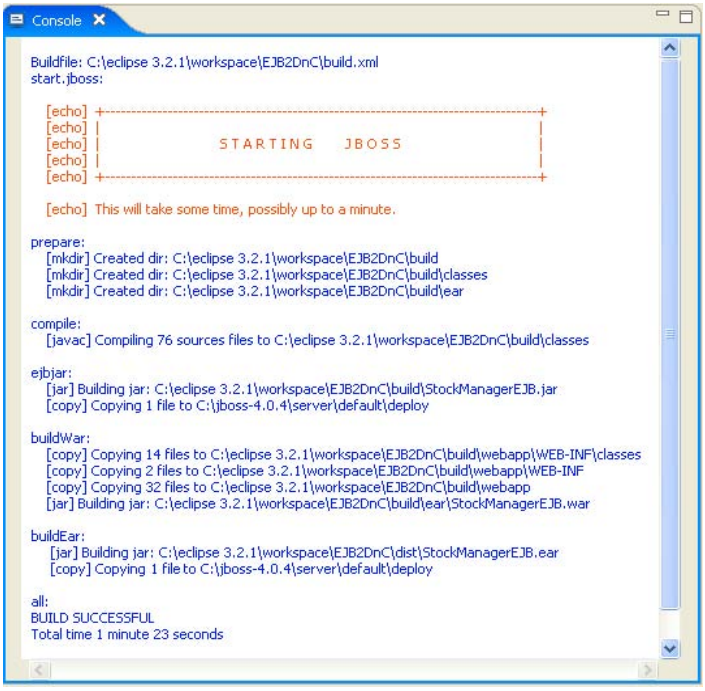

Fig. 13: Result of deployment plan running

- Start running the JBoss server
- Preparing and compiling Java Bean classes
- Packaging and deploying the result files

The output screenshot of this file is presented in Fig. 13.

CONCLUSION AND PERSPECTIVES

Software deployment is a complex process. It covers the post-development activities such as release, install, configure, plan, launch, de-install, de-release. Many component technologies (e.g., CCM, EJB, .Net) exist. On the other hand, a generic model that covers all these technologies would be desirable. The main contribution of this study is to present a generic deployment approach using D and C specification and MDA technique. The proposed approach is tested over EJB model and is implemented such as:

- Transformation of EJB PSM to D and C application model PIM by means of ATL tool
- Execution of deployment plan, that describes an assembly of component instance by specifying where, they are distributed and 'how' they are connected and configured

Further work remains to be done on several aspects. Our future plans consist in two areas:

- Extension of the system, thus tries to investigate the integration of other component systems (e.g., CCM, .Net) and novel application architecture such as Service Oriented Architecture (SOA) and embedded systems
- Intention to evolve the prototype with dynamic (re) configuration of components

REFERENCES

Akehurst, D.H. and S.J.H. Kent, 2002. A relational approach to defining transformations in a metamodel. Proceedings of 5th International Conference on the Unified Modeling Language, Dresden, Germany, 30 September-October 4, Springer Berlin/Heidelberg, pp: 243-258.

Bézivin, J., F. Jouault, P. Rosenthal and P. Valduriez, 2005. Modeling in the large and modeling in the small. Proceedings of European MDA Workshops: Foundations and Applications, MDAFA 2003 and MDAFA 2004, Twente, The Netherlands, June 26-27, Springer Berlin/Heidelberg, pp: 33-46.

Bézivin, J. and F. Jouault, 2006. Using ATL for checking models. Proceedings of International Workshop on Graph and Model Transformation, GraMoT 2005, March 27, Tallinn, Estonia, pp: 69-81.

Carzaniga, A., A. Fuggetta, R.S. Hall, D. Heimbigner and A. Van der Hoek *et al*., 1998. A characterization framework for software deployment technologies. Technical Report CU-CS-857-98. Department of computer Science University of Colorado, USA.

Deng, G., J. Balasubramanian, W. Otte, D.C. Schmidt and A. Gokhale, 2005. DAnCE: A QoS-enabled component deployment and configuration engine. Proceedings of 3rd Working Conference on Component Deployment, CD 2005, Grenoble, France, November 28-29, Springer Berlin/Heidelberg, pp: 67-82.

Flissi, A. and P. Merle, 2004. Vers un environnement multi-personnalités pour la configuration et le déploiement des applications à base de composants logiciels. Proceedings of 1st French Conference on Software Deployment and (Re) Configuration, DECOR'04, October 28-29, Grenoble, France, pp: 3-14.

Gerber, A., M. Lawley, K. Raymond, J. Steel and A. Wood, 2002. Transformation, the missing link of MDA. Proceedings of 1st International Conference, ICGT 2002, Barcelona, Spain, October 7-12, Springer Berlin/Heidelberg, pp: 90-105.

Gray, J., Y. Lin and J. Zhang, 2006. Automating change evolution in model-driven engineering. Special issue on Model-Driven Engineering. IEEE Comput. Soc., 39: 51-58.

Hnětynka, P., 2004. Making deployment of distributed component-based software unified. Proceedings of CSSE 2004 (Part of ASE 2004), Linz, Austria, September 20-24, Austrian Computer Society, pp: 157-161.

Jouault, F. and I. Kurtev, 2005. Transforming models with ATL. Proceedings of 5th International Conference on Model-Driven Engineering Languages and Systems, MoDELS 2005. Montego Bay, Jamaica, October 2-7, Springer Berlin/Heidelberg, pp: 128-138.

Kebbal, D. and G. Bernard, 2001. Component search service and deployment of distributed applications. Proceedings of 3rd International Symposium on Distributed Objects and Applications, DOA 01, September 17-20, Roma, Italy, pp: 125-134.

Lestideau, V. and N. Belkhatir, 2003. Providing highly automated and generic means for software deployment process. Proceedings of 9th International Workshop on Software Process Technology, EWSPT 2003, Helsinki, Finland, September 1-2, Springer Berlin/Heidelberg, pp: 128-142.

Merle, N. and N. Belkhatir, 2004. Open architecture for building large scale deployment systems. Proceedings of International Conference on Software Engineering Research and Practice, SERP '04, June 21-24, Las Vegas, Nevada, USA., pp: 930-936.

Monson-Haefel, R. and B. Burke, 2006. Enterprise Java Beans 3.0. 5th Edn., O'Reilly Media, Inc., Cambridge, MA, USA., ISBN-10: 0-596-00978-X, pp: 760.

OMG, 2004. Deployment and configuration of component-based distributed applications specification. http://comquad.inf.tudresden.de/nfc04/presentations/OMG%20D&C%20Spec%20(Francis%20Bordeleau).pdf.

Quema, V., R. Balter and L. Bellisard, 2004. Asynchronous, hierarchical, and scalable deployment of component-based applications. Proceedings of 2nd International Working Conference on Component Deployment, CD 2004. Edinburgh, UK, May 20-21, Springer Berlin/Heidelberg, pp: 50-64.

Varró, D., G. Varró and A. Pataricza, 2002. Designing the automatic transformation of visual languages. Sci. Comput. Programm., 44: 205-227.

Wang, A.J.A. and K. Qian, 2005. Component-Oriented Programming. 1st Edn., John Wiley and Sons Inc., Chichester, West Sussex, UK., ISBN 13: 9780471713708.