UDeploy: A Unified Deployment Environment

Mariam Dibo and Noureddine Belkhatir

Laboratoire d'Informatique de Grenoble
681, Rue de la Passerelle, BP 72, 38402, St. Martin d'Hères, France
{Mariam.Dibo,Noureddine.Belkhatir}@imag.fr

Abstract. In the software life cycle we have mainly three activities: (1) the pre-development (requirements, specification and design), (2) the development (implementation, prototyping, testing) and (3) the post-development (deployment). Software deployment encompasses all post-development activities that make an application operational. These activities, identified as deployment life cycle, include: i) software packaging, ii) loading and installation of software on client sites, iii) instance creation, iv) configuration and v) updating. The development of system-based components made it possible in order to highlight this part of the global software lifecycle, as illustrated by numerous industrial and academic studies. However these are generally developed ad hoc, and consequently platform-dependent. Deployment systems, such as supported by middleware environments (CCM, .Net and EJB), specifically develop mechanisms and tools related to pre-specified deployment strategies. Our work, related to the topic of distributed component-based software applications, aims at specifying a generic deployment framework independent of the target environments. Driven by the meta-model approach, we first describe the abstractions used to characterize the deployed software. Then, we specify the deployment infrastructure and processes, highlighting the activities to be carried out and the support for their execution.

Keywords: Deployment, Meta model, Model, Software component, MDA.

1 Introduction

Component-based software approach [25] is intended to improve the reuse of component enabling the development of new applications by assembling pre-existing components. A software component can be deployed independently and may be composed by third parties [25].

Nowadays, the component approach and distribution make deployment a very complex process. Many deployment tools exist, we identified three types of systems: 1) those developed by the industry and integrated into a middleware environment like EJB [8], CCM [21] and .Net [26, 27]; 2) those projected by the OMG (industry) [22] [9] based on more generic models and; 3) the more formal systems projected by academic works in current component models like Open Service Gateway Initiative (OSGI) [1], Web Services [11], SOFA [3], Architecture Description Languages (ADL) [4] and UML 2.0 [24].

Generally, deployment tools are often built in an ad hoc way; i.e. specific to a technology or an architecture and covering partially the deployment life cycle (using generally the installation scripts).

Hence, deployment is seen as the post development activities that make software usable. It covers the description of the application to deploy, the description of the physical infrastructure, the description of the deployment strategies, the planning activities and the plan execution.

The deployment issue deals with aspects as diverse as satisfying software and hardware constraints of the components concerning the resources of the machines that support them, the resolution of inter-component dependency, the installation and "instantiation" of components via the middleware and the container, the interconnection of components, their activation and the management of dynamic updates. Thus, the challenge [5] is to develop a generic framework encompassing a specific approach and supporting the whole deployment process. [6] presents the conceptual framework of this approach and [7] presents the different models based on the MDA approach [23].

This paper focuses on the implementation part fulfilled by UDeploy (models transformation) and the presentation of a case study to illustrate our approach. The rest of this paper is organized as follows: part 2 presents the related works. Part 3 presents the architecture of our deployment tool. Part 4 presents the model transformation. Finally in part 5, we present the perspectives of this work.

2 Related Works

We identified several works on the deployment that have been classified into two broad categories.

In the first category, there are mainly all the more classic works developed for the monolithic software systems and that emphasize on the setup activity.

In the second category, there are all the systems of deployment developed recently for the software based-components. We identified two types of systems in this category:

- those developed by industry on an ad 'hoc way and integrated into a middleware type of environments;
- those of a higher level of abstraction based on explicit model proposed by the OMG on one hand and on the other hand by the academic world.

2.1 Deployment in Middleware

The pros of deployment in application based-component like EJB [8], CCM [21] and .Net [26, 27] relay in the fact that the technologies are effective thus answers specific needs. The cons are that the abstraction level is very low therefore it is necessary to make each activity manually. In such contexts and with these facts, it is easy to deduce that there is a real need to standardize the deployment of distributed applications. The middleware does not support the description of the domain. They contain less semantics to describe applications; for example, the needs of an application may be a specific version of software, and a memory size greater than 10 GB. Since none of these constraints will be checked during installation, this corresponds to a single copy component assembly. The deployment descriptor expresses the same mechanism for each middleware but described them in different ways.

2.2 Deployment in OMG Specification

The industry felt the necessity to join their efforts. They anticipated an approach which capitalizes on their experiences in deployment (OMG's approach). This specification has inspired many academics. OMG's Deployment and Configuration (D&C) [22] specification is based on the use of models, meta-models and their transformation. This specification standardizes many aspects of deployment for component-based distributed systems, including component assembly, component packaging, package configuration, and target domain resource management. These aspects are handled via a data model and a runtime model. The data model can be used to define/generate XML schemas for storing and interchanging metadata that describes component assemblies and their configuration and deployment characteristics. The runtime model defines a set of managers that process the metadata described in the data model during system deployment. An implementation of this specification is DAnCE (Deployment And Configuration Engine) [9].

2.3 Deployment in Academic Approaches

In current component models like, Open Service Gateway Initiative (OSGI) [1], Web Services [11], SOFA [3], Architecture Description Languages (ADL) [4] and UML 2.0 [24], components are defined in the form of architectural units [15]. The ADL [19] such as Acme, AADL, Darwin and Wright allow modeling components, to model connectors and to model architecture configurations; however deployment process in ADL is not specified. UML2.0 allows describing system hardware. But deployment diagram in UML2.0 is a static view of the run-time configuration of processing nodes and the components that run on those nodes. Other approaches such as SOFA do not address the processing part. The plan containing the information on the application is directly executed from a centralized server, assuming that remote sites can instantiate remote components from this server.

Tables 1, 2, 3 and 4 presented in annex, present an assessment related to three main notions occurring in the constitution of a deployment system which are the application, the domain, the deployment strategies and the deployment plan.

- The domain notion covers all machines connected to a network where a software system is deployed. This infrastructure is seen as a set of distributed and interconnected sites. Each site is associated with the meta-information of the site characteristics descriptions.
- The application notion covers all the application components and the meta-information for their descriptions.
- The deployment strategies guide the creation of the deployment plan. The deployment strategies allow expressing the actions to be led to deploy a component by assuring success and safety properties.
- The deployment plan for an application A consists of components C1 to Ci where i>= 1 and for a domain D consisting of Sites S1 to Sj where j> = 1 is all valid

placements (Ci, Sj). It is calculated from a planner engine. This engine operates on a static process which allow visualizing a state of the system and the information remains motionless during the computing plan or following a dynamic process which allows visualizing the forecasts and to supervise their realization; the information used is variable during the computing plan.

3 UDeploy Architecture

Concerning to the assessment obtained from the state of the art practice of the related works, we think that a good solution to automate component based systems deployment owes to [6]:

- cover all deployment activities,
- be independent from technologies,
- be independent from any philosophy of components based approach,
- offer a distributed deployment engine,
- propose specific language strategies in order to make the deployment flexible and to support existing strategies in the deployment environments.

The analysis of a deployment system highlights activities independent from technologies and what we could factor as the:

- modeling of the application to deploy,
- modeling of the components execution environment,
- creation of the deployment plan.

Therefore, we propose a deployment architecture [7] based on MDA (Model-Driven Architecture) approach [23] with the use of models, meta-models and their transformation (MDA approach is described in the section 4.1). MDA approach allows offering a unified framework based on deployment activities using generic descriptors that may subsequently be customized for specific platforms.

Deployment study in enterprise business practices allowed us to understand that the deployment must be flexible according to the needs of the company and according to the technical specifications of the application. Hence, we propose a fourth meta-model related to deployment strategies in addition to the three common meta-models.

Figure 1 illustrates this deployment process comprising the following six main activities:

- The **application modeling** which describes the application to be deployed; in other words, it specifies all the components that compose the application and, the resource constraints of these components.
- The **domain modeling** which describes the deployment environment, meaning which specifies all sites that compose it and the available resources.
- The deployment strategies modeling which allow describing the policies to be implemented in order to make the deployment plan flexible according to specific needs.
- The creation of the deployment plan which from an application model, a domain model and a deployment strategies model produce a deployment plan.

- The transformation covers two main activities:
  - the customization of the deployment plan - the deployment plan produced at the end of the deployment plan creation activity is at a pim level (platform independent model), therefore it is independent from any technology. this deployment plan is seen as a set of placements. this generic plan must be customized to one or several psm level plans (platform specific model); i.e. specific to technologies so that they can be executed by the middleware targets. the deployment plan answers to the question "where to deploy?".
  - the generation of the deployment descriptor - the deployment descriptor is built from information within the application model and also from other information (application non-functional properties) produced by the *deployer*. the deployment descriptor answers to the question "how the container must manage components to deploy?".

- The deployment plan execution - some middleware do not offer any support for the implementation of the deployment plan. in that case, the generic plan will be translated into an appropriate description of the target middleware (script). This description will be carried out by our deployment tool by invoking methods of the target middleware.

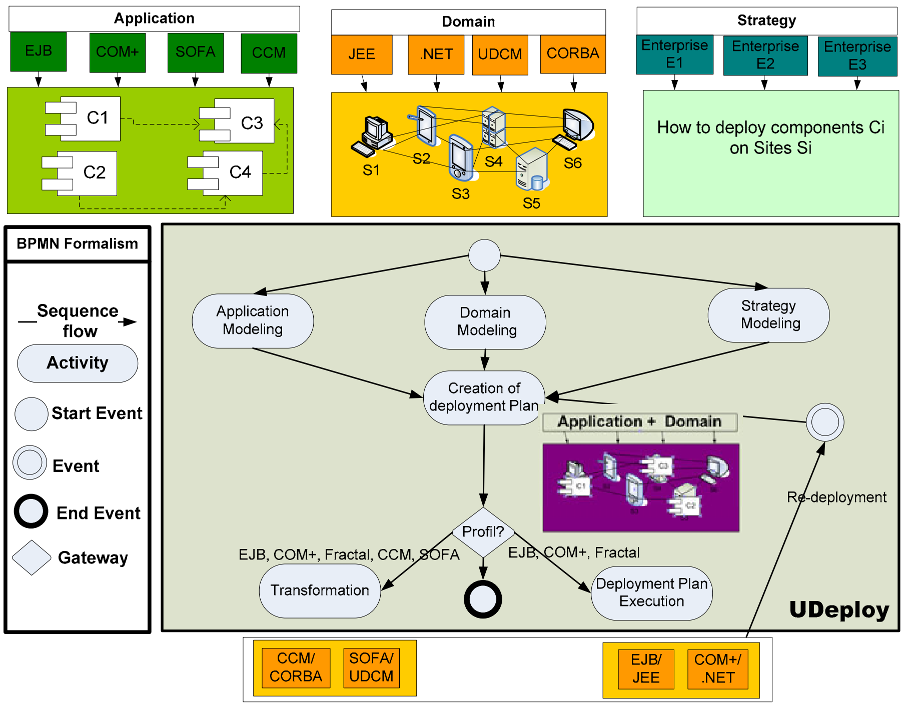


Fig. 1. UDeploy architecture

4 Model Transformation

4.1 MDA Approach

The MDA approach [23] has been proposed by the OMG in response to the problems posed by the multiplicity of systems, languages and technologies. The main idea of the MDA approach is the separation of technical concerns from trades [10]. The key concepts inherent to the MDA approach are:

- The PIM (Platform Independent Model) - these models are independent on the technology platforms such as EJB, CCM, COM + and, provide a high level of abstraction.
- The PSM (Platform Specific Model) - these models are dependent on the technology platforms and correspond to the executable code.
- The transformation - PIM to PSM or PSM to PIM passage occurs by models transformations. A model transformation is defined from a set of rules. These rules can be described using a QVT type transformation tool (Query View Transformation) [20], or by implementing its own processing tool. There are several transformations tools and languages such as QVT-core (MTF [12]), QVT-relations (medini QVT [18]) and QVT-like ATL [14, 13], Tefkat [17] and VIATRA [28]).

4.2 MDA Advantage

The main advantages of the MDA approach are productivity and portability [16]. Productivity is because developers can now focus on the development of the PIM models. They will work at a level where technical details are no longer specified. These technical details will be added to the PSM level at the time of processing. This improves productivity in two ways. First and foremost, PIM developers will omit specific details. Second, several PSM can be obtained for different platforms with less effort. Portability is because a PIM may be automatically transformed into several PSM for various platforms. Thus, everything specified at PIM level will remain portable. The only thing needed is to make sure that the code to be generated is conform to the technology of an execution target platform.

4.3 MDA and Deployment

Conventional deployment tools integrated into the middleware, re-develop in a specific manner the mechanisms and the deployment processes. These tools can be seen to be at the PSM level. So, applying MDA to deploy would define deployment meta-models at PIM level and that can be customized for different platforms.

4.4 Transformation Language

Transformation of models [2] may be operated by a non-formal language, by a specific QVT or by a transformation algorithm that sets the mapping between different models. The transformation language that we propose is mixed, hence based on the QVT ATL and on transformation algorithms (figure 2).

Model transformation is not based on the UDeploy application model, the domain model and the UDeploy strategies model, but covers the UDeploy deployment plan and the UDeploy deployment descriptor model.

Transformation of the deployment plan model consists of the projection of the UDeploy plan model from a PIM level to a PSM level plan models (EJB, CCM, .NET, SOFA). Specific deployment plan models are executed by middleware targets in order to implement the deployment.

Transformation of the deployment descriptor model consists of the transformation of the UDeploy descriptor model from a PSM level to a PIM level descriptor models (EJB, CCM). Specific deployment descriptor models are used by the middleware targets to manage components.

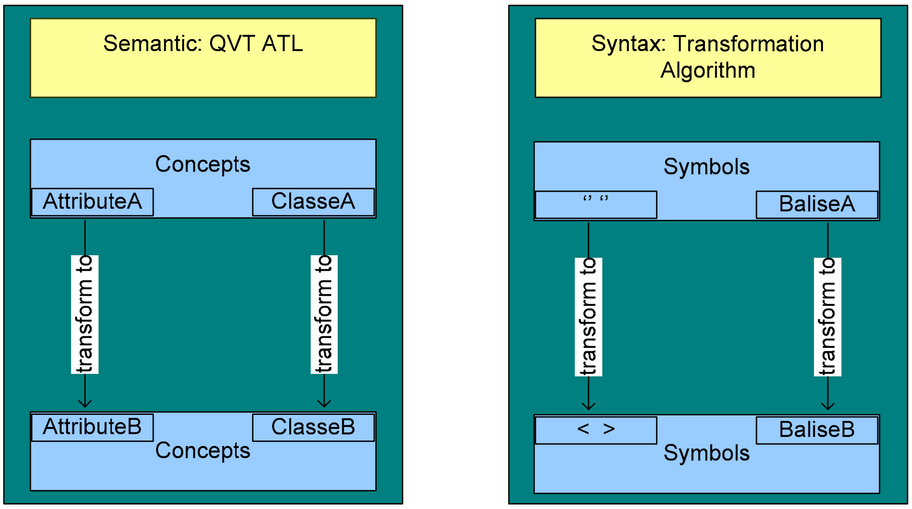


Fig. 2. Transformation language (QVT ATL and algorithm)

4.5 QVT ATL

We use the QVT ATL for semantic transformation (Figure 3). Semantic transformation corresponds to the transformation of the concepts. A concept A in a source model might be called concept B in a target model. ATL is a model transformation language developed on top of the Eclipse platform. It provides ways to generate target models from source models via transformation rules. An ATL transformation rule is written as follow:

```
rule R {
  from e : source-meta-model ! el-e (cond)
  to s : target-meta-model ! el-s
  (-- ex. title<- e.title, name<- e.name+ "new")
}
```

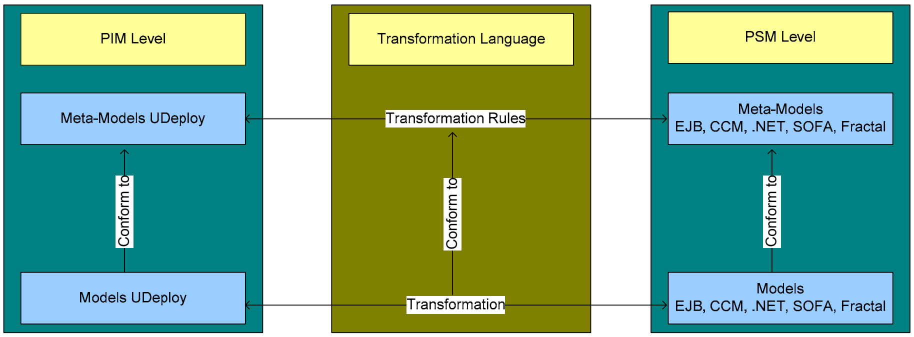


Fig. 3. Transformation QVT ATL

4.6 Transformation Algorithm

We use algorithms for syntactic transformation. An M1 model that meets a source meta-model criteria might be written in Java while an M2 model compliant to a target meta-model might be written in XML. The UDeploy deployment plan meta-models and the UDeploy deployment descriptor meta-models are written in DTD (Document Type Definition). For practical reasons, we have decided to develop our algorithms and to manage the models' persistence with Java. Hence, we needed to operate three basic transformations (figure 4):

- The transformation of the DTD UDeploy meta-models to XSD UDeploy meta-models via the XMLPad tool.
- The transformation of the XSD UDeploy meta-models to Ecore UDeploy meta-models via the EMF tool.
- The transformation of the Ecore UDeploy meta-models to Java UDeploy meta-models via the EMF tool.

The chain of transformation from the DTD meta-model plan and the DTD meta-model descriptor to the Java meta-model plan and the Java meta-model descriptor does occur only once.

Once the Java classes are created, they will be instantiated by the deployment plan data and the deployment descriptor.

We have syntactic transformation (Figure 4) for each technology such as EJB (AlgoEJBPlan, AlgoEJBDescriptor algorithms CCM AlgoCCMDescriptor (AlgoCCMPlan), .NET (AlgoNETPlan) and SOFA (AlgoSOFAPlan). The algorithm allows producing a target model which will be conformed syntactically to the target meta-model.

4.7 Examples of Model Transformation

4.7.1 EJB, NET and CCM Deployment Plan Personalization (Semantic)

At the end of the planning process, we obtain a PIM level UDeploy deployment plan model. This deployment plan must be customized for execution target platforms.

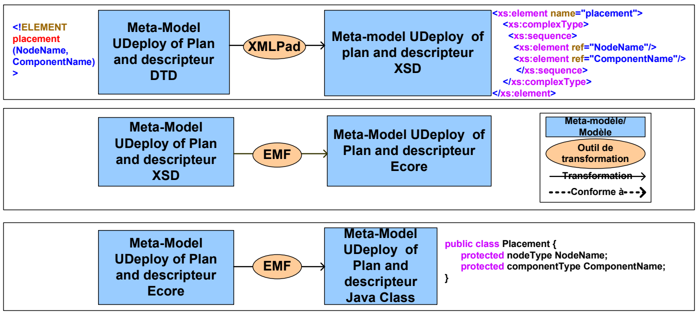

<!ELEMENT placement (NodeName, ComponentName) >
Meta-Model UDeploy of Plan and descripteur DTD
XMLPad
Meta-model UDeploy of plan and descripteur XSD
<xs:element name="placement">
<xs:complexType>
<xs:sequence>
<xs:element ref="NodeName"/>
<xs:element ref="ComponentName"/>
</xs:sequence>
</xs:complexType>
</xs:element>
Meta-Model UDeploy of Plan and descripteur XSD
EMF
Meta-Model UDeploy of Plan and descripteur Ecore
Meta-modèle/ Modèle
Outil de transformation
Transformation
Conforme à
Meta-Model UDeploy of Plan and descripteur Ecore
EMF
Meta-Model UDeploy of Plan and descripteur Java Class
public class Placement {
protected nodeType NodeName;
protected componentType ComponentName;
}

Fig. 4. Transformation algorithme

The example below shows the transformation process of the UDeploy deployment plan meta-model to the EJB, .NET and CCM platforms plan meta-model.

The rule #1 takes as input the UDeploy deployment plan meta-model (source) and as output an EJB, .NET and CCM deployment plan meta-model (target). The transformation concerns the *DeploymentPlan* class of the source meta-model and the *DeploymentPlan class* of the target meta-model. The *PlanId* attribute of the target meta-model will be the *PlanId* attribute of the source meta-model.

```
rule R1 {
from in : UDeployDeploymentPlanMetaModel ! DeploymentPlan
to out : EJB_NET_CCMDeploymentPlanMetaModel ! DeploymentPlan
PlanId<- in.PlanId }
```

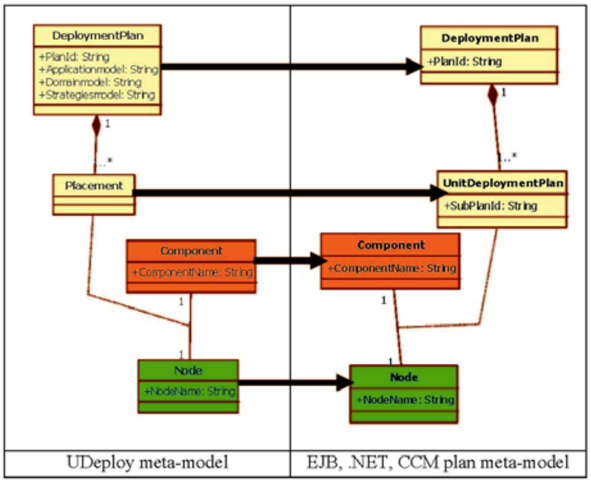


Fig. 5. Semantic transformation

The rule #2 takes as input the UDeploy deployment plan meta-model (source) and as output an EJB, .NET and CCM deployment plan meta-model (target).

The transformation concerns the DeploymentPlan class of the source meta-model and the *UnitDeploymentPlan class* of the target meta-model.

The Sub*PlanId* attribute of the UnitDeploymentPlan class will be a concatenation of the the *PlanId* attribute of the source model and a plan number supplied by the user (getSubPlanNmber () method).

```
    rule R2 {
    from in : UDeployDeploymentPlanMetaModel ! DeploymentPlan
    to out : EJB_NET_CCMDeploymentPlanMetaModel ! UnitDeploymentPlan
    SubPlanId<- in.PlanId + getSubPlanNmber()}
```

The rule #3 takes as input the UDeploy deployment plan meta-model (source) and as output an EJB, .NET and CCM deployment plan meta-model (target).

The transformation concerns the *component* class of the source meta-model and the *component class* of the target meta-model. The *ComponentName* attribute of the target meta-model will be the *ComponentName* attribute of the source meta-model.

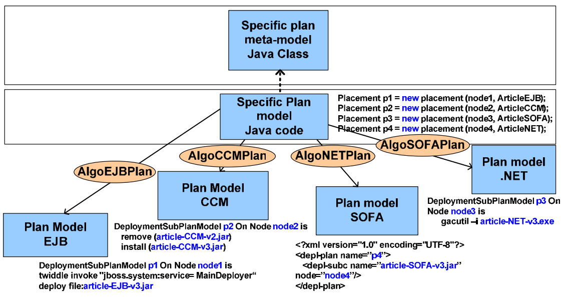
```
    rule R3 {
    from in : UDeployDeploymentPlanMetaModel ! Component
    to out : EJB_NET_CCMDeploymentPlanMetaModel ! Component
    ComponentName<- in.ComponentName }
```

The rule #4 takes as input the UDeploy deployment plan meta-model (source) and as output an EJB, .NET and CCM deployment plan meta-model (target). The transformation concerns the Node class of the source meta-model and the the *Node* class of the target meta-model. The *NodeName* attribute of the target meta-model will be the *NodeName* attribute of the source meta-model.

```
    rule R4 {
    from in : UDeployDeploymentPlanMetaModel ! Node
    to out : EJB_NET_CCMDeploymentPlanMetaModel ! Node
    NodeName<- in.NodeName }
```

4.7.2 EJB, .NET and CCM Deployment Plan Customization (Syntactic)

Below, we will present four examples of syntactic customization (Figure 6). The customization algorithms of the deployment plan for the EJB, CCM, .NET and SOFA platforms are respectively AlgoEJBPlan, AlgoCCMPlan, AlgoNETPlan and AlgoSOFAPlan.



```
Placement p1 = new placement (node1, ArticleEJB);
Placement p2 = new placement (node2, ArticleCCM);
Placement p3 = new placement (node3, ArticleSOFA);
Placement p4 = new placement (node4, ArticleNET);
```

```
DeploymentSubPlanModel p3 On
Node node3 is
    gacutil –i article-NET-v3.exe
```

```
DeploymentSubPlanModel p2 On Node node2 is
  remove (article-CCM-v2.jar)
  install (article-CCM-v3.jar)
```

```
<?xml version="1.0" encoding="UTF-8"?>
<depl-plan name=''p4''>
  <depl-subc name=''article-SOFA-v3.jar''
node=''node4''/>
</depl-plan>
```

```
DeploymentSubPlanModel p1 On Node node1 is
twiddle invoke "jboss.system:service= MainDeployer"
deploy file:article-EJB-v3.jar
```

Fig. 6. Syntactic transformation

```
AlgoEJBPlan
Input: Specific deployment plan EJB mEJB
Ouput: document d
Debprog;
       document d;
       For each placement p in mEJB do
              C=getComponentName(p);
              IC=getImplementation(C);
              N=getNodeName(p);
              NT=getNodeServerType(N);
              if (NT== JBOOS) then d.write('On Node', N , 'is
              twiddle invoke "jboss.system:service= MainDeployer"
              deploy file:',IC);
              endif;

              else if (NT==JONAS) then d.write('On Node', N ,
              'jonas admin -a',IC);
              endelseif ;
       endo;
       Return d;
Finprog;
```

```
AlgoNETPlan
Input: Specific deployment plan .NET mNET
Ouput: document d
Debprog;
       document d;
       For each placement p in mNET do
              C=getComponentName(p);
              IC=getImplementation(C);
              N=getNodeName(p) ;
              d.write('On Node', N, ' is gacutil -i',IC);
       endo;
       Return d;
Finprog;
```

```
AlgoCCMPlan
Input: Specific deployment plan CCM mCCM
Ouput: document d
Debprog;
       document d;
       For each placement p in mCCM do
              C=getComponentName(p);
              IC=getImplementation(C);
              N=getNodeName(p);
              d.write('On Node', N, 'Install(',IC, ')');
       enddo;
       Return d;
Finprog;
```

5 Conclusions and Perspectives

We develop UDeploy, a prototype based on the MDA approach which ensures tree main tasks: (i) it manages the planning process from meta-information related to the application, the infrastructure and the deployment strategies, (ii) it generates specific deployment descriptors related to the application and the environment (i.e. the machines connected to a network where a software system is deployed) and (iii) it executes a deployment plan.

We have positive feedbacks with our case study and its experimentation on EJB, .NET and CCM platforms. Our current projects include carrying out other experiments and evaluations to show the feasibility of the approach, for example its application to industrial systems, .NET and CCM.

References

1. Alliance, O.: OSGi 4.0 release. Specification (October 2005), `http://www.osgi.org/`
2. Bézivin, J., Büttner, F., Gogolla, M., Jouault, F., Kurtev, I., Lindow, A.: Model transformations? transformation models! In: Wang, J., Whittle, J., Harel, D., Reggio, G. (eds.) MoDELS 2006. LNCS, vol. 4199, pp. 440–453. Springer, Heidelberg (2006)
3. Bures, T., Hnetynka, P., Plasil, F.: Sofa 2.0: Balancing advanced features in a hierarchical component model. In: SERA, pp. 40–48. IEEE Computer Society, Los Alamitos (2006)
4. Clements, P.C.: A survey of architecture description languages. In: IWSSD 1996: Proceedings of the 8th International Workshop on Software Specification and Design, p. 16. IEEE Computer Society, Washington, DC, USA (1996)
5. Dibo, M., Belkhatir, N.: Challenges and perspectives in the deployment of distributed components-based software. In: ICEIS (3), pp. 403–406 (2009)
6. Dibo, M., Belkhatir, N.: Defining an unified meta modeling architecture for deployment of distributed components-based software applications. In: 12th International Conference on Enterprise Information Systems (ICEIS), Funchal, Madeira, Portugal (June 2010)
7. Dibo, M., Belkhatir, N.: Model-driven deployment of distributed components-based software. In: 5th International Conference on Software and Data Technologies (ICSOFT), Athens, Greece (July 2010)
8. Dochez, J.: Jsr 88: Java enterprise edition 5 deployment api specification (2009), `http://jcp.org/aboutJava/communityprocess/mrel/jsr088/index.html`
9. Edwards, G.T., Deng, G., Schmidt, D.C., Gokhale, A.S., Natarajan, B.: Model-driven configuration and deployment of component middleware publish/subscribe services. In: Karsai, G., Visser, E. (eds.) GPCE 2004. LNCS, vol. 3286, pp. 337–360. Springer, Heidelberg (2004)
10. Gerber, A., Lawley, M., Raymond, K., Steel, J., Wood, A.: Transformation: The missing link of mda, pp. 90–105. Springer, Heidelberg (2002)
11. Gustavo, A., Fabio, C., Harumi, K., Vijay, M.: Web Services: Concepts, Architecture and Applications (2004)
12. IBM. Mtf: Model transformation framework (2010), `http://www.alphaworks.ibm.com/tech/mtf`
13. Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I.: Atl: A model transformation tool. Sci. Comput. Program. 72(1-2), 31–39 (2008)

14. Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I., Valduriez, P.: Atl: a qvt-like transformation language. In: OOPSLA Companion, pp. 719–720 (2006)
15. Kaur, K., Singh, H.: Evaluating an evolving software component: case of internal design. SIGSOFT Softw. Eng. Notes 34(4), 1–4 (2009)
16. Kleppe, A.G., Warmer, J., Bast, W.: MDA Explained: The Model Driven Architecture: Practice and Promise. Addison-Wesley Longman Publishing Co., Inc., Boston (2003)
17. Lawley, M., Steel, J.: Practical declarative model transformation with tefkat. In: MoDELS Satellite Events, pp. 139–150 (2005)
18. mediniQVT. medini qvt (2010), `http://projects.ikv.de/qvt`
19. Medvidovic, N., Taylor, R.N.: A classification and comparison framework for software architecture description languages. IEEE Trans. Softw. Eng. 26(1), 70–93 (2000)
20. OMG. MOF QVT Final Adopted Specification. Object Modeling Group (June 2005)
21. OMG. Corba component model 4.0. (2006), specification `http://www.omg.org/docs/formal/06-04-01.pdf`
22. OMG. Deployment and configuration of component-based distributed application (2006), specification `http://www.omg.org`
23. T.O.M.G. OMG. Omg model driven architecture (2005), `http://www.omg.org`
24. T.O.M.G. OMG. Unified modeling language (2007), `http://www.omg.org`
25. Szyperski, C., Gruntz, D., Murer, S.: Component Software: Beyond Object-Oriented Programming, 2nd edn. Addison-Wesley Professional, England (2002)
26. Troelsen, A.: Chapter 1: The Philosophy of .NET, vol. Pro VB 2008 and the .NET 3.5 Platform. APress (2008)
27. Troelsen, A.: Chapter 15: Introducing.NET Assemblies, vol. Pro VB 2008 and the.NET 3.5 Platform. APress (2008)
28. Varró, D., Balogh, A.: The model transformation language of the viatra2 framework. Sci. Comput. Program. 68(3), 214–234 (2007)

Appendix

Table 1. Application meta-model comparison

Approach	Application meta-model			
	Software architecture	Software constraints	Hardware constraints	Descriptor Format
EJB	*	/	/	Conform to DTD ejb-jar
CCM	*	*	*	Conform to DTD SoftwarePackageDescriptor.dtd CORBAComponentDescriptor.dtd
.Net	*	* (only assembly dependencis)	/	Manifest MSI
D&C	*	*	*	ComponentDataModel ComponentManagementModel
Software Dock	*	*	*	Conform to DTD DSD
Orya	*	*	*	Product model
Fractal	*	*	*	Fractal ADL (xml)
SOFA	*	/	/	SOFA component meta-model
UML	*	*	*	Component diagram

* (supported) / (no-supported)

Table 2. Domain meta-model comparison

Approach	Domain meta-model			
	Hardware architecture	Software resources	Hardware resources	Descriptor Format
EJB	/	/	/	/
CCM	/	/	/	/
.Net	/	/	/	
D&C	*	*	*	TargetDataModel TargetManagement Model
Software Dock	*	*	*	Fieldock Releasedock
Orya	*	*	*	Site model
Fractal	/	/	/	/
SOFA	* Docks (remote node)	/	/	Sofanode (centralized node)
UML	*	*	*	Deployment diagram

* (supported) / (no-supported)

Table 3. Deployment strategies meta-model comparison

Approach	Deployment strategies meta-model			
	Technology	Enterprise	Fixed/ Flexible	Language for stratégies specification
EJB	*	/	Fixed	/
CCM	*	/	Fixed	SoftwarePackageDescriptor.dtd CORBAComponentDescriptor.dtd CORBAassemblyDescriptor.dtd
.Net	*	/	Fixed	*(only for application update)
D&C	/	/	/	/
Software Dock	*(configuration)	/	Fixed	/
Orya		* (few semantic)	Flexible	Strategies model
Fractal	*		Fixed	
SOFA	*		Fixed	* (only for dynamic adaptation via DCUP)
UML	/	/	/	/

* (supported) / (no-supported)

Table 4. Deployment plan meta-model comparison

Approach	Deployment plan meta-model			
	Processus de planification supporté	Plan de déploiement complet	Plan de déploiement exécutable	Format du plan de déploiement
EJB	/	/	/	Script
CCM	/	/	/	Script
.Net	/	/	/	Script
D&C	*	*	*	XML document for CCM/Dance
Software Dock	*	*	/	Embedded in the tool (code)
Orya	*	*	/	Embedded in the tool (code)
Fractal	/	/	/	/
SOFA	/	*	*	XML Document
UML	/	/	/	Deployment Diagram

* (supported) / (no-supported)