Applying MDA Approach for Web Service Platform

Jean Bézivin[(1)] Slimane Hammoudi[(2)] Denivaldo Lopes[(1)(2)] Frédéric Jouault[(1)(3)]

[(1)] Atlas Group, INRIA and LINA
University of Nantes
2, rue de la Houssinière - BP 92208
44322 Nantes Cedex 3, France

[(2)] ESEO
4, rue Merlet de la Boulaye, BP 926
49009 cedex 01 Angers, France

[(3)] TNI-Valiosys
120, rue René Descartes
Technopôle Brest Iroise - BP 70801
29608 Brest Cedex, France

{Jean.Bezivin, Frederic.Jouault}@lina.univ-nantes.fr {shammoudi, dlopes}@eseo.fr

Abstract

In this paper, we present the development of an illustrative example of e-business based on two different applications of a Model-Driven Architecture (MDA) approach. In the first application, the Platform-Independent Model (PIM) is created using the Unified Modeling Language (UML). This PIM is transformed using Atlas Transformation Language (ATL) to generate the Platform-Specific Model (PSM) based on three target platforms: Java, Web Service and Java Web Service Developer Pack (JWSDP). In the second application, the PIM is created using Enterprise Distributed Object Computing (EDOC) and transformed into another PSM based on the same target platforms. For this purpose we will use the illustrative example of a travel agency to depict some issues of the approach. Then, we will discuss mappings between meta-models.

1. Introduction

The central problem facing software developers at the beginning of this century is the handling of systems that are much more complex than before. They are complex not only because the volume of code and data involved is much larger but for many other reasons as well:

- They have to integrate a lot of new aspects (e.g. security, reliability and performance) from the start, in a uniform way;
- These systems evolve much more quickly than previously. This is not only because business and application requirements change, but also because the underlying technological platforms are constantly evolving, at a very fast rate. Protecting software investments from obsolescence due to platform integration is a very important goal;
- As new technology arrives at a fast rate, old technologies do not disappear, but are concentrated in the legacy software. As a consequence the evolution problem is augmented by a heterogeneity problem.

These three factors combined create a situation that may be perceived as probably the biggest crisis until now in the software industry.

In order to find reasonable solutions to these problems, a radical paradigm shift seems necessary. The object technology introduced in the 80's has given all that it could and does not seem in a position to give much more. The component technology has not significantly improved this situation and may be viewed as adding additional complexity to a domain that needs simplification.

So the main proposed solution to the new software crisis seems to be a paradigm change from object composition to model transformation. Will that be sufficient? For the time being it seems there are few alternative ways.

The idea is thus to consider everything as a model or a model element. Models represent aspects of a system and they can be combined. Models may also be transformed into other models and the transformation itself is a model. With this approach it is possible to separate platform independent aspects in Platform-Independent Models (PIMs) from platform specific aspects in Platform-Specific Models (PSMs).

The general idea seems then quite simple: use model transformation to generate PSM from PIM. In practice we still have a lot to learn before this may be generally applied in industrial environments. The goal of this work is to bring additional insight from the PIM to PSM transformation process.

Models are not only unconstrained labeled oriented graphs. Each model is based on the precise vocabulary and additional properties of its unique meta-model. One can easily understand that there is not a unique meta-model for PIM and another unique meta-model for PSM. Determining the meta-models that will be used in PIM to PSM transformation is a hard task. We shall proceed here by practical experimentation, trying to draw the conclusions of our work in the last part of this paper.

We have chosen the travel agency problem as an illustrative example to discuss some issues of the Model Driven Architecture (MDA) approach to develop e-business. This illustrative example is modeled in two ways: first, using the Unified Modeling Language (UML) and secondly the Enterprise Distributed Object Component (EDOC) [15]. According to MDA concepts, both models are considered as PIMs that are manipulated and transformed following specific rules to create their PSMs based on Java, Web Service and Java Web Service Developer Pack (JWSDP). We have used the Atlas Transformation Language (ATL) [2] to define the transformation rules.

This paper is organized as follows. In section 2, we present an overview of MDA and Web Service, and we situate our research. In section 3, the illustrative example is introduced through use case diagram. In section 4, we show the PIM of a travel agency created with UML, and the steps of transformation from this PIM to PSM based on Java, Web Service and JWSDP. In section 5, EDOC is used to build the PIM of the illustrative example, and the steps of transformation from EDOC to Java, Web Service and JWSDP are presented. In the last section, we analyze our results and discuss the future direction of our research and approach.

2. Background

In recent years, MDA and Web Service have been introduced and developed by organisms of standardization, Object Management Group (OMG) and World Wide Web Consortium (W3C), respectively. MDA consists of a set of standards that assists the creation, implementation, evolution and deployment of systems driven by models. In the context of MDA, Web Service is only one of many possibilities for target models [6][9]. Other target models can also be used with MDA as Enterprise JavaBeans (EJB) or CORBA Component Model (CCM) [1].

2.1. MDA

The OMG's Model Driven Architecture (MDA) separates the modeling task of the implementation details, without losing the integration of the model and the development in a target platform. The key technologies of MDA are Unified Modeling Language (UML), Meta-Object Facility (MOF), XML Meta-Data Interchange (XMI) and Common Warehouse Metamodel (CWM) [14]. Together, they unify and simplify the modeling, the design, the implementation, and the integration of systems. One of the main ideas of MDA is that each model is based on a specific meta-model. Each meta-model precisely defines a domain specific language. Finally, all meta-models are based on a meta-meta-model. In the MDA technological space, this is the Meta-Object Facility (MOF). There are also standard projections on other technological spaces like XMI for projection on XML and Java Metadata Interface (JMI) for projection on Java [8].

MDA also introduces other important concepts: Platform-Independent Model (PIM), Platform-Specific Model (PSM), transformation language, transformation rules and transformation engine. These elements of MDA are depicted in figure 1.

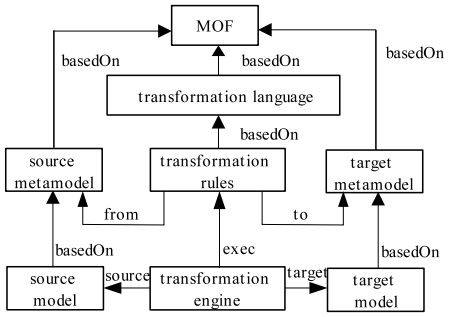

Figure 1. Transformation in MDA

Each element presented in Figure 1 plays an important role in MDA. In our approach, MOF is the well-established meta-meta-model used to build meta-models. The PIM reflects the functionalities, the structure and the behavior of a system. The PSM is more implementation-oriented and corresponds to a first binding phase of a given PIM to a given execution platform. The PSM is not the final implementation, but has enough information to generate interface files, a programming language code, an interface definition language, configuration files and other details of implementation. Mapping [4] from PIM to PSM determines the equivalent elements between two meta-models. Two or more elements of different meta-models are equivalents if they are compatible and they cannot contradict each other. Model transformation is realized by a transformation engine that executes transformation rules. Transformation rules specify how to generate a target model (i.e. PSM) from a source model (i.e. PIM). To transform a given model into another model, the transformation rules map the source into the target meta-model. The transformation engine takes the source model, executes the transformation rules, and gives the target model as output.

Using one unique formalism (e.g. MOF) to express all meta-models is very important because this allows the expression of all sorts of correspondence between models based on separate meta-models. Transformations are one important example of such correspondence, but there are also others like traceability, etc. In other words, given $m_1(s)/M_a$ and $m_2(s)/M_b$, where m_1 is a model of a system s created using the meta-model M_a, and m_2 is a model of the same system s created

using the meta-model M_b, then a transformation can be defined as $m_1(s)/M_a \rightarrow m_2(s)/M_b$. When M_a and M_b are based on the same meta-meta-model, the transformation may be expressed in a unique transformation language (i.e. a language independent of the meta-model). Furthermore, the transformation language itself may be considered as a domain-specific language. This has many interesting consequences. One of these is that a transformation program corresponds to an MDA model. We may thus easily consider higher-order transformations, i.e. transformations having other transformations as input and/or producing other transformations as output.

One of the most popular meta-models is UML, but there are plenty of other meta-models being defined. For example, there could be a meta-model of the Java language. Based on these two meta-models, it is possible to express a transformation from UML 1.5 class diagrams to Java 1.4.2 code. In fact, models have been used for a long time, but they remained disconnected from the implementation process.

The automatic generation of code to a specific language from a UML class diagram is not new either; some CASE tools give this support such as Poseidon for UML (http://www.gentleware.com). However developers still have to write all the application codes by hand. Moreover, when the application has to evolve to acquire new capabilities or adapt to new technologies, these tools cannot help the developer, and the model is used only as documentation.

An Integrated Development Environment (IDE) provides a set of tools integrated on a single user interface that often comprises a sophisticated text editor, a graphical editor for GUI, an editor to database tables, a compiler and a debugger, e.g. IBM's WebSphere Studio and Microsoft's Visual Studio .NET. An IDE can aid the software development, but only in the programming phase (i.e. it is based on code-centric approach).

A tool powered with MDA will be enabled to support the system development throughout its life cycle. The development of large software systems can take some suggested benefits (some benefits are still not proven) from MDA:

- the same PIM can be used many times to generate models on different platforms (PSMs) [7];
- many views of the same system, e.g. many abstraction levels or details of implementation [14]. We define abstraction levels as the possibility to see a system (e.g. applications and business process) fragmented in many different and interlinked levels, each level detaching important characteristics of the same system;
- enhancement of the portability and of the interoperability of systems in the level of models;
- preservation of the business's logic against the changes or evolution of technology [14];
- a uniform manner for business models and for technologies to evolve together;
- prevention against error-prone factors linked to manual development of systems [12];
- increase the return on technology investments;
- enhancement of the reengineering [3], i.e. it assists the recuperation of business's logic from source codes or from implementation environments;
- enhancement of the interaction and of the migration between different technological spaces [3].

Apart from these benefits, the approach using models forces the architects to think about the architecture and the model behind the system in development, whereas a code-centric approach makes architects concentrated on the code, so they consequently forget the main properties of the system.

Other case studies have shown some benefits of the MDA Approach. In [12], the authors have demonstrated that the development of a case study (i.e. J2EE PetStore) using a MDA tool is 35% faster than the development using a code-centric approach (i.e. using a non powered-MDA tool).

2.2. Web Service

A Web Service is a software application that has its interfaces, its bindings and its invocations defined, described, and discovered using XML artifacts. The key technologies that constitute a Web Service are: eXtensible Markup Language (XML); Simple Object Access Protocol (SOAP) [18]; Web Service Description Language (WSDL) [19]; and Universal Description, Discovery and Integration (UDDI) [17]. XML is an extensible markup language that has been used for document and data representation. It is a simple and powerful solution for the problem of Electronic Data Interchange (EDI). SOAP is a protocol to exchange information in decentralized and distributed systems. UDDI is the universal register of Web Services and its core is based on XML files that may store information about a business entity and its services. WSDL is an abstract definition based on XML grammar to describe the syntax and semantics necessary to call up a service.

Other researches have considered some issues in the evolution of Web Services, e.g. performance, interoperability, usability and composition [11] [5] [16]. However, few of them consider the MDA approach [6].

3. The Illustrative Example

To study the issues of the MDA approach, we have chosen the illustrative example of a travel agency.

A travel agency sells flight tickets, reserves hotel rooms and provides car rental. In order to provide these services

for its customers, a travel agency needs to establish business links with other enterprises, i.e. airlines, car rental companies and hotels. In this context, a financial institution, i.e. a bank, is required to facilitate the financial transactions between a customer and a business or between a business and another business. Figure 2 presents a simplified use case diagram for a travel agency and its partners.

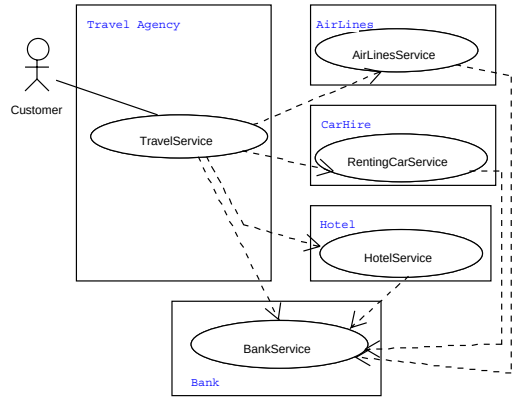

Figure 2. Use case diagram of the Travel Agency

Figure 2 shows a customer who plans to make a trip. In order to make this trip, he accesses the Web site of a travel agency that sells flight tickets, provides car rental and room reservations. The customer enters his requirements. The travel agency receives the requirements of the customer and sends them to the airline, car-hire company, and hotel. The travel agency receives the possibilities from the three partners and choses the best solutions for flights, cars and hotels. It sends them to the customer who choses, reserves and pays for a flight ticket, a car and a hotel room. The payment is made with the support of a bank.

In the next sections, this illustrative example will be modeled, implemented, and deployed, driven by models using the MDA approach. Firstly, the PIM of this illustrative example will be created using UML, and the PSM will be generated on Java, Web Service and JWSDP. In the second experiment, the PIM of the same illustrative example will be based on EDOC that will be transformed into a PSM, based on the same target platforms.

4. The PIM using UML

Figure 3 shows the PIM of the illustrative example. In this paper, the fragment of the PIM is presented without the behavioral modeling part to simplify the presentation of our work. In this figure, six classes implement the main characteristics of the system in our illustrative example: `Customer`, `TravelAg`, `AirLines`, `CarHire`, `Hotel` and `Bank`.

The `Customer` accesses the travel agency service. The `TravelAg` provides the services to find, reserve, pay and cancel a flight ticket, a car and a room. The `AirLines`, `CarHire` and `Hotel` provide services to find, reserve, pay and cancel their products. The `Bank` relays the financial transaction among the partners.

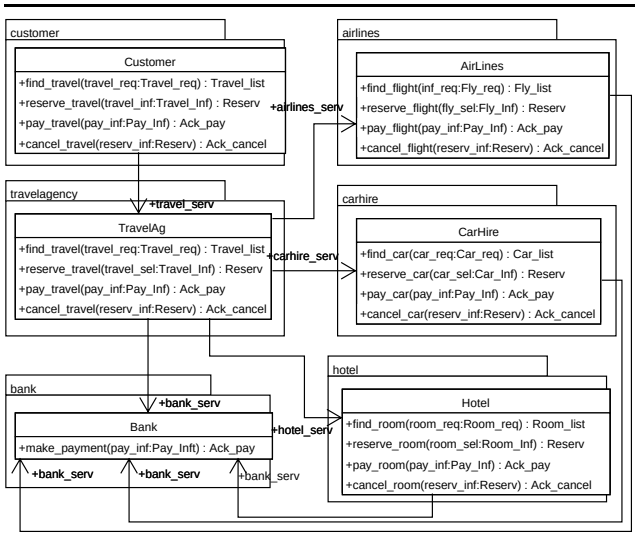

Figure 3. PIM of Case Study Travel Agency(fragment)

4.1. From PIM (UML) to PSM (Java)

Figure 1 depicts the steps of model transformations in the context of MDA. According to this approach, the source and target meta-models must be instances of the MOF. The source meta-model chosen was UML that is MOF compliant. The PIM is an instance of UML, then it is MOF compliant too. In order to make the transformation, the PSM must be created as an instance of a meta-model based on MOF. As the target platforms are Java, Web Service and Java Web Service Developer Pack (JWSDP), their MOF compliant meta-models are required.

Figure 4 presents a possible meta-model for the Java language (fragment). The main elements of this Java meta-model are:

- `JavaElement` - is a generalization for other elements such as `JavaPackageElement`, `JavaMember`, `JavaValue` and `JavaParameter`;
- `JavaPackageElement` - is a generalization for the `JavaPackage`;

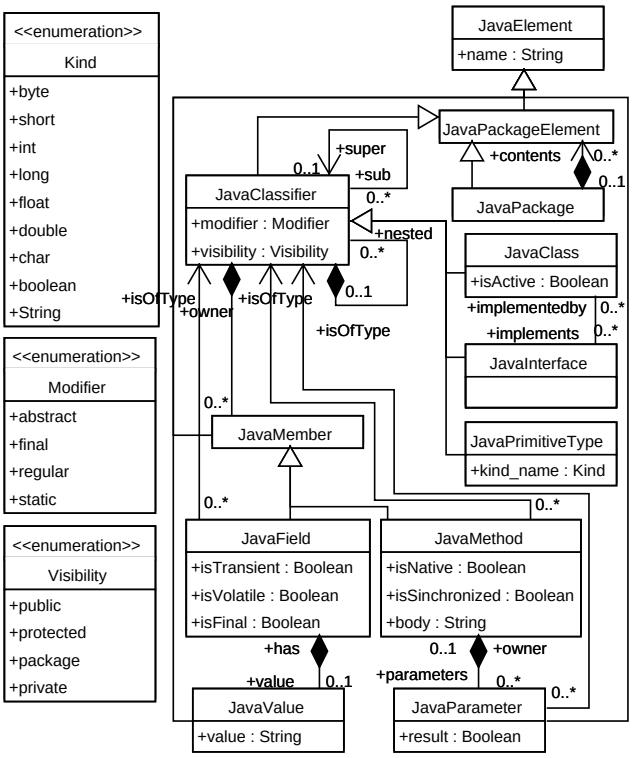

Figure 4. Java Meta-model

- `JavaPackage` - is a container for all classes and interfaces of Java;
- `JavaClassifier` - is the base for `JavaPrimitiveType`, `JavaClass` and `JavaInterface`. It defines the association `super` for inheritance and has `JavaMember` (i.e. the base for `JavaField` and `JavaMethod`);
- `JavaPrimitiveType` - can be a `byte`, `short`, `boolean`, ...;
- `JavaClass` - is a specialization of `JavaClassifier` that has `JavaField` and `JavaMethod` as member;
- `JavaInterface` - is a specialization of `JavaClassifier` and it has the prototypes of methods (i.e. signature or contract) and constant `JavaFields` (i.e. `final static attributes`);
- `JavaField` - is a composition of only one value and it has a type `JavaPrimitiveType`, `JavaClass` or `JavaInterface`;
- `JavaMethod` - has the signature of the operation, the `JavaParameters` and the `body` of the method;
- `JavaParameter` - is the argument of the `JavaMethod` and it can be input (if `result` is `false`) or output (if `result` is `true`);
- `JavaValue` - is used to initialize a `JavaField` with an initial `value`.

The model transformation needs to determine what elements of the UML meta-model will be mapped to the elements of the Java meta-model.

Figure 5 shows a possible mapping from the UML meta-model to the Java meta-model (i.e. fragment). This mapping is realized by a set of rules that specify the elements of the source meta-model, i.e. UML, which are equivalents to elements of the target meta-model, i.e. Java meta-model. In this figure, the mapping of two meta-models is presented using a graphical notation. The UML meta-model is presented in the left side, the mapping in the center, and the Java meta-model in the right side. This graphical notation has the following elements: connection (source and target), association, transformation rule and composition. A connection links one or more meta-model element(s) to a transformation rule. The association shows a relation between rules. The composition shows a tight relation between rules (i.e. composite rules). The transformation rule takes a source element and generates the suitable target element. In our work, we have used ATL [2] to create transformation rules, but this can be made using eXtensible Stylesheet Language Transformation (XSLT) [10] or logical languages [7] or other transformation language based on Object Constraint Language (OCL) as ATL.

XSLT and logical languages make possible the manipulation of XML files (i.e. structured documents) to accomplish the transformation from source to target model. Both are efficient for models based on simple meta-models, but they are limited and error-prone for the transformation of models based on large meta-models.

ATL[1] is MDA compliant and uses a repository to store and to manipulate the source and target meta-models in order to perform the transformation following the mapping defined using transformation rules. In this work, we have used MDR repository [13]. It stores metadata as Java and/or CORBA objects that are exposed using Java and/or CORBA interfaces.

Despite the use of XSLT and logic languages to make transformation rules, ATL is simpler and allows manipulations to be closer to the elements of meta-models and models. In fact, ATL was designed to make model transformation, and XSLT was designed to make general transformation.

After identifying the equivalent elements between UML and Java meta-models, we can describe the mappings from UML to Java. Thus, we have the following mappings (we will present only a few ATL transformation rules):

- The `UML Class` is mapped to `JavaClass` through the rule `C2C`:

1 A new implementation of ATL under Eclipse is scheduled for 2004.

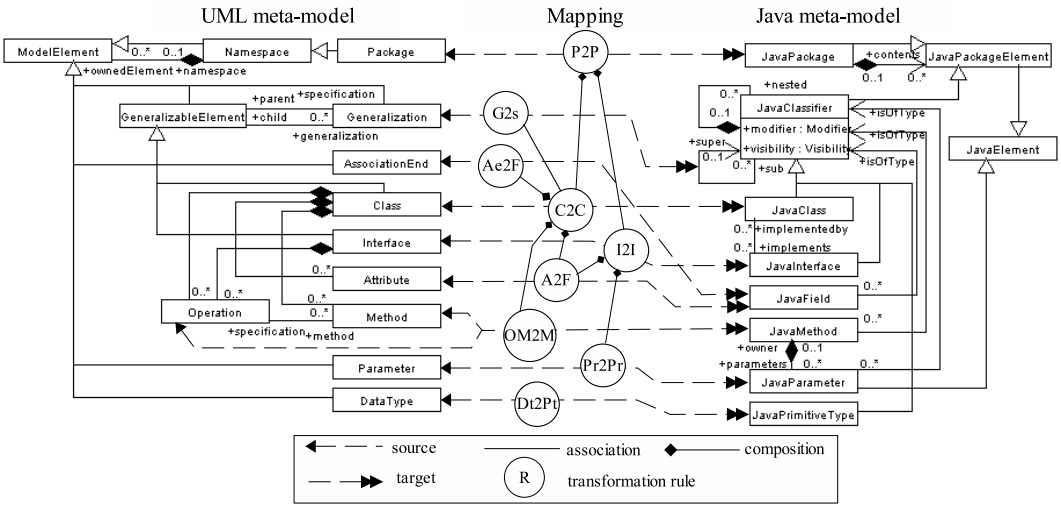

Figure 5. Mapping from UML to Java Meta-model (fragment)

```
rule C2C{
from c : UML!Class
to jc : Java!JavaClass
mapsTo c(
 name <- c.name,
 visibility <-
 if c.visibility = #vk_public then
  #public
 else
  #private
 endif,
 modifier <-
  if c.isAbstract then
   #abstract
  else if c.isLeaf then
   #final
  else
   #regular
  endif endif,
 isActive <- c.isActive,
 super <-
  c.generalization->first().parent,
 implements <-
  c.clientDependency->
    select(e|e.hasSterotype('realize'))
    ->collect(e | e.supplier))
}
```

This rule creates an instance of `Java!JavaClass` named `jc` and initializes it with the same characteristics as the `UML!Class c` (from the source model). Thus, name is given the value of `c.name`, `visibility` is set according to the value of `c.visibility`, `modifier` is set according to the value of `c.isAbstract` or `c.isLeaf`. The attribute `isActive` is given the value of the `c.isActive`, `super` is assigned with a `Java!JavaClass` (corresponding to `c`'s `parent`). The implements is set to a `Java!JavaInterface` (corresponding to `UML!Interface` implemented by `c`). Since we are working here with Java and we want to simplify our discussion about model transformation, we will not allow UML's multiple inheritance. However, the UML's multiple inheritance can be handled in Java meta-model using Java Interfaces.

- The `UML AssociationEnd` is mapped to `JavaField` through the rule `Ae2F`:

```
rule Ae2F{
  from ae : UML!AssociationEnd
  to  jf : Java!JavaField
  mapsTo ae(
  name <- ae.name,
  visibility <-
   if ae.visibility = #vk_public then
    #public
   else
    #private
   endif,
  isTransient <- false,
  isVolatile <- false,
  isFinal <- false,
  isOfType <- ae.participant,
  owner <- ae.getOtherEnd().participant)
}
```

This rule creates an instance of `Java!JavaField` and initializes it with the same characteristics as `UML!AssociationEnd`. Thus, `name` is given the value of `ae.name`, `visibility` is set according to the value of `ae.visibility`. The attribute `isTransiend`, `isVolatile` and `isFinal` are set to `false`. The attribute `isOfType` is assigned with a `ae.participant`. The `owner` is set according to `Java!JavaClass` or `Java!JavaInterface` (`const field`).

The mapping from UML meta-model to Java meta-model also shows the interdependencies between rules. Rule `P2P` involves calling the rules `C2C` and `I2I`. Rule `C2C` involves calling the rules `Ae2F`, `OM2M` and `A2F`. Rule `I2I` involves calling the rules `OM2M` and `A2F` (`constant field`).

Figure 5 shows the existence of many-to-one mappings. A many-to-one mapping has many elements of the meta-model source and only one element of the meta-model target. This is a consequence of the semantic difference

between the source and target meta-model. Sometimes, the semantics of two or more elements of a meta-model is represented using only one element of another meta-model. In the case of the mapping from UML to Java meta-model, the `UML Operation` and `Method` are mapped into `JavaMethod`.

4.2. From PIM (UML) to PSM (Web Service)

As shown in Figure 1, the transformation of PIM (UML) into PSM (Web Service) requires the support of two meta-models: source and target. The source is UML and so no further commentary is needed. On the other hand, we need to provide specific meta-models for Web Services.

Among Web Service technologies, the most important for our study are UDDI and WSDL. However, to simplify our discussion about Web Service in the context of MDA, we have chosen only to present the WSDL meta-model and the mapping from UML meta-model to WSDL meta-model. Figure 6 shows the WSDL meta-model.

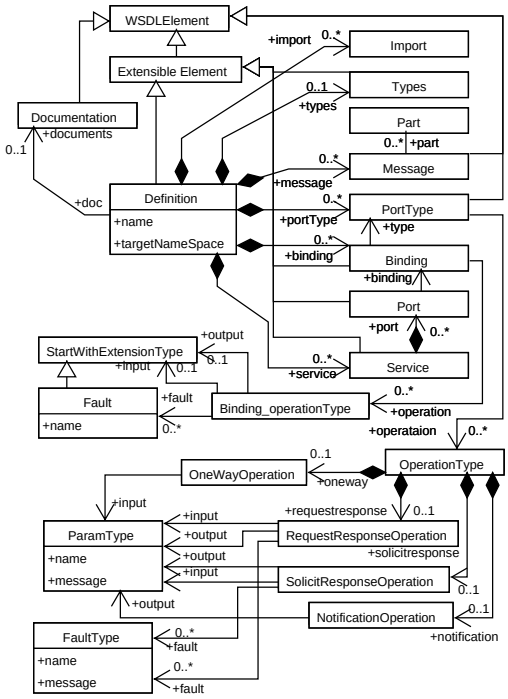

Figure 6. WSDL Meta-model

WSDL is basically composed of:

- *Types* - describes abstract data types;
- *Message* - describes the message structure;
- *PortType* - presents the Web Service Interface;
- *Binding* - represents the way the service is accessed;
- *Service* - describes who provides the service.

The transformation engine (see Figure 1) gets the UML model and obtains the WSDL instance. In the transformation process, rules map from UML meta-model instances to WSDL meta-model instances.

Figure 7 depicts the mapping from UML meta-model to WSDL meta-model. According to this figure, we have the following mappings:

- The `UML Interface` is mapped to `WSDL PortType`, `Binding` and `Service` through the rule `I2Pt`:

```
rule I2Pt{
from itf : UML!Interface
to pt : WSDL!PortType
mapsTo itf(
  name <- itf.name,
  operations <- [O2O.wsdlop]
  itf.feature ->
    select(e | e.oclIsKindOf(UML!Operation))),
bd : WSDL!Binding(
  name <- itf.name + 'Binding',
  type <- pt,
  operations <- [O2O.wsdlob]
  itf.feature ->
  select(e | e.oclIsKindOf(UML!Operation))),
sv : WSDL!Service(
  name <- 'Service' + itf.name,
  port <- pport),
pport : WSDL!Port(
  name <- itf.name+'Port',
  binding <- bd,
  soap <- ssoap),
ssoap : WSDL!SOAP(
  location <-
    'http://host:port/'  +
    context-path/url-pattern ' -- to be replaced)
}
```

  This rule creates an instance of `WSDL!PortType`. The `name` is set according to the value of `itf.name`. The `operations` is assigned with all `wsdlop` (instance of `WSDL!PortTypeOperation`) created according to the `UML!Operation` owned by the current `UML!Interface`. It also creates an instance of `WSDL!Binding`. The `name` is set with the value of `itf.name + 'Binding'`. The `type` is `pt` (i.e. the created `WSDL!PortType`). The `operations` is assigned with all `WSDL!BindingOperation wsdlob` created according to the `UML!Operation` owned by the current `UML!Interface`. It also creates `WSDL!Service` and initializes it.

Figure 7 shows the existence of one-to-many mapping. A one-to-many mapping has one element of the meta-model source and many elements of the meta-model target. This is also a consequence of the semantic difference between the source and target meta-model. Sometimes, the semantics of one element of a meta-model is represented using many elements of another meta-model. In the case of the mapping from UML to WSDL meta-model, the `UML Interface` is mapped into `WSDL PortType`, `Binding` and `Service`.

As shown in this section, one benefit of MDA is its capability to allow the generation of several implementations, on different platforms, from a business model. In other words,

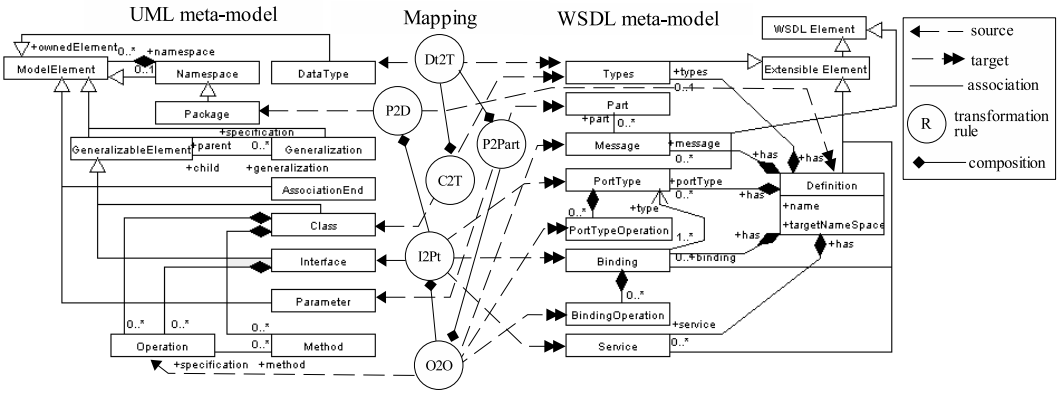

Figure 7. Mapping from UML to WSDL Meta-model (fragment)

the business system characteristics are only acquired and modeled once.

4.3. JWSDP platform

In sections 4.1 and 4.2, we have shown a transformation from PIM into PSM for our illustrative example. To create the final implementation of the example, it is still necessary to define a JWSDP meta-model, detailing the deployment files.

An important characteristic of JWSDP is the manner in which a service is implemented. A `JavaClass` that provides its methods as services must implement an Interface that extends `java.rmi.Remote` and its methods throw `java.rmi.RemoteException`. Figure 8 shows the simplified JWSDP meta-model.

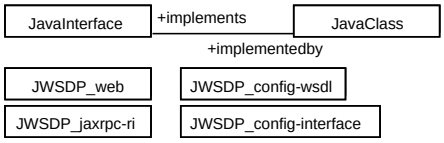

Figure 8. JWSDP Meta-model (fragment)

JWSDP needs four deployment files: `web.xml`, `jaxrpc-ri.xml`, `config-wsdl.xml` and `config--interface`. In our approach, these files are created using the information contained in `JavaClass`. These generated deployment files are incomplete and must be finalized before the deployment. In fact, deployment files have information that is available only in the time of deployment.

For example, the `JavaClass` is mapped to `JavaClass`, `JavaInterface`, `JWSDP_web`, `JWSDP_jaxrpc_ri`, `JWSDP_config_wsld` and `JWSDP_config_interface` through an ATL transformation rule. This rule creates a `JavaClass` and a `JavaInterface`. The `JavaClass` will have the same characteristics as the original `JavaClass`, but it will implement the generated `JavaInterface`. It also creates the deployment files.

4.4. The PSM based on Java, Web Service and JWSDP

In the transformation process, we proceeded as follows. First, we built the UML model of the travel agency (section 3 and 4). Second, we showed how a UML model can be transformed into a Java model (section 4.1). Second, we depicted how a UML model can be transformed into a WSDL model (section 4.2). Third, we presented how a Java model can be transformed into JWSDP model (section 4.3).

The PSM shown in Figure 9 was generated from the PIM of the travel agency. It is an instance of the meta-models of Java, Web Service and JWSDP represented as UML profiles. We used UML profiles only to visualize the final PSM, and all model transformations were based on specific meta-models. In this figure, some details are omitted to simplify the presentation of the target model. Car hire and hotel services are not shown, but they have the same structure as the airline.

This PSM is not yet the final implementation, but has enough information to generate part of the final code and deployment files. The PSM is a model that can be exported outside of the MDR repository as a XMI file or as Java files and description files (i.e. XML files). Afterwards, the Java files and description files must be completed.

This application of the MDA approach illustrates how a formal model of a business (i.e. PIM of travel agency) can be transformed into several platforms (PSMs).

Our experience creating transformation rules has revealed that the use of UML profiles to develop a PIM and build its PSM has some disadvantages: large transformation rules; excessive utilization of filters and instructions of flow control (if-then-else); excessive comparison of strings. Using profiles, we cannot profit from the main advantages of a language such as ATL, i.e. a transformation language

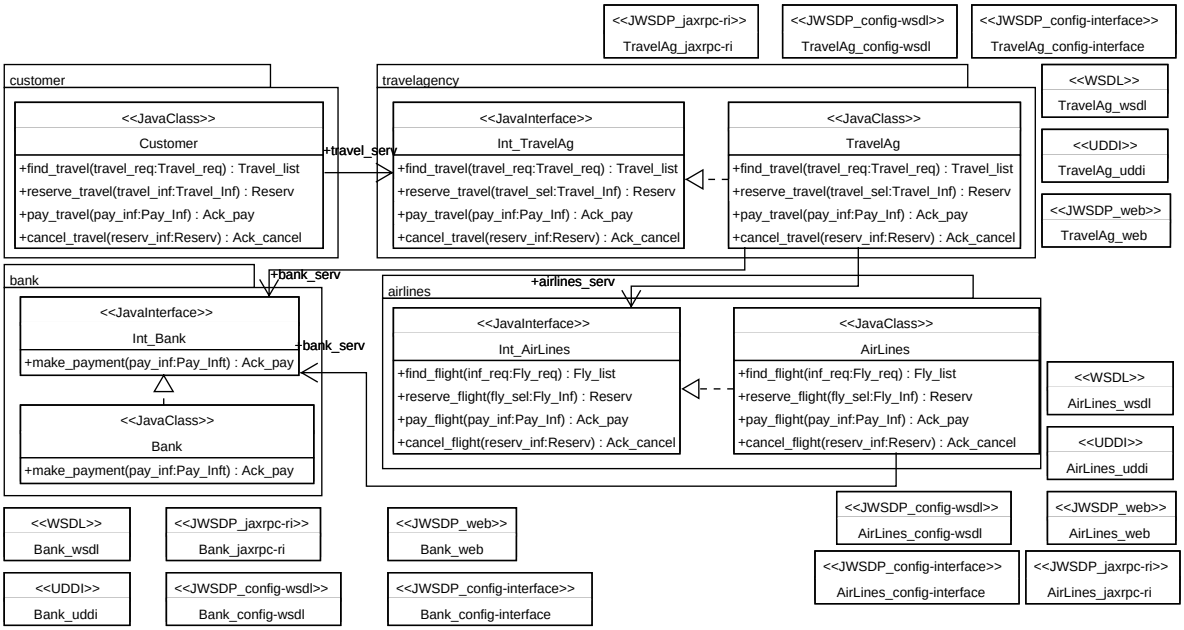

Figure 9. PSM of the Case Study Travel Agency(fragment)

designed to manipulate meta-models and models. The use of profiles to build PIMs and PSMs makes the creation of transformation rules difficult. On the other hand, the manipulation of specific meta-models based on MOF have given some benefits such as simplicity of the transformation rules, transformation rules can directly manipulate meta-model elements and short transformation rules.

5. The PIM using EDOC Model

The UML profile for EDOC is a framework to develop components based on EDOC systems [15]. It is composed of a set of profiles which define: Enterprise Collaboration Architecture (ECA), patterns, and technology specific models and technology mappings.

The ECA allows the definition of PIMs and provides five UML profiles:

- *Component Collaboration Architecture (CCA)* uses classes, activity and collaboration graphs to model the structure and behavior of components;
- *Entity profile* uses a set of UML extensions to model entity objects;
- *Events profile* is a set of UML extensions to model the events of a system;
- *Business Process profile* complements the CCA and models the system behavior;
- *Relationships profile* extends the UML core facilities to attend to the requirements of the relationship in general and business modeling and software modeling.

Patterns provide standard models that can be employed to elaborate good object models for EDOC systems.

The ECA is focused on the specification of the structural and behavioral parts of EDOCs. On the other hand, the *technology specific models* and the *technology mappings* are part of EDOC specification that takes into account the mapping from ECA specification to technology specific models. It defines an EDOC profile for Enterprise Java Beans and another for Flow Composition Model (FCM).

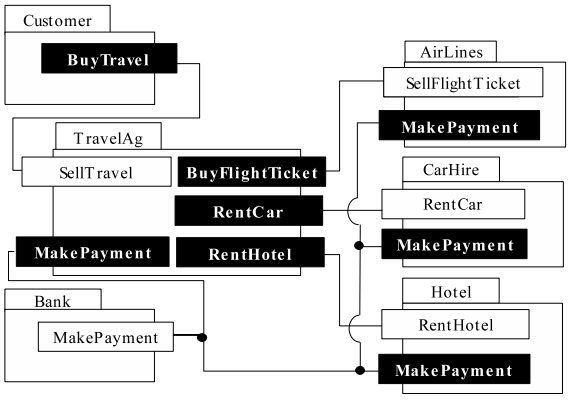

Figure 10. The community Process of the Travel Agency (CCA Notation)

Figure 10 shows the community process for our illustrative example using the CCA notation. It is a simplified model that only shows the components that participate in the business process. The travel community process speci-

fies how a customer, a travel agency, airline, car-hire enterprise, hotel, and bank collaborate to complete a transaction within a business. Each role is played by the suitable component usage, i.e., `Customer`, `TravelAg`, ..., `Bank`.

The `Customer` collaborates directly with the `TravelAg` through the `BuyTravel` and `SellTravel` protocol ports, according to the `BuySellTravel` protocol. The `TravelAg` collaborates with the `AirLines` through the `BuyFlightTicket` and `SellFlightTicket` protocol ports, according to the `BuySellFlight` protocol. As the `AirLines`, the `TravelAg` collaborates with the `CarHire`, `Hotel` and `Bank` to accomplish a transaction. Figure 11 presents the contracts of components using the UML profiles for EDOC (i.e. UML notation).

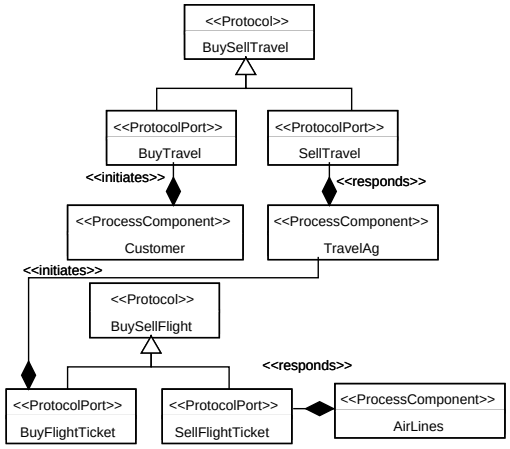

Figure 11. Contracts of Components

The activities in the community process Travel Agency start by the `Customer` initiating the interactions on its `BuyTravel` protocol port, according to the `BuySellTravel` protocol presented in Figure 11 and further depicted in Figure 12 that details the structure. The structure is composed of other protocols, such as `FindTravel`, `ReserveTravel`, ..., `CancelTravel`. The `TravelAg` will follow the `BuySellTravel` protocol, and when opportune will initiate the `BuySellFlight`,..., and `BuySellHotel` protocol.

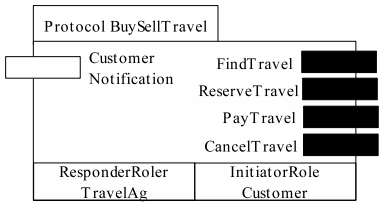

Figure 12. BuySellTravel Protocol structure

5.1. From PIM (EDOC) to PSM (Java)

Figure 13 depicts the mapping from the EDOC-CCA meta-model [15] to the Java meta-model. The main elements of the EDOC-CCA meta-model used in our approach are:

- `DataElement` - a generalization for `DataType` or `CompositeData` (a type of data);
- `FlowPort` - extends `Port` and describes the form which may produce or consume a single `DataElement`;
- `OperationPort` - defines a port which implements the typical call-return semantics and one-way operation;
- `ProtocolPort` - defines the use of a protocol and can involve the use of two-way interactions between components;
- `ProcessComponent` - a processing component that collaborates with other `ProcessComponents` using Ports within a CCA composition. It externalizes compositional objects as a component.

Before creating the mapping between the EDOC-CCA and Java meta-model, we must find the equivalent elements. In figure 13, `DataType` is equivalent to `JavaPrimitiveType`, `CompositeData` is equivalent to `JavaClass`, `Attribute` is equivalent to `JavaField`, and so on.

After identifying the equivalent elements between EDOC-CCA and Java meta-models, we can describe the mapping from EDOC to Java (we will present only a few ATL transformation rules):

- `CompositeData` is mapped to `JavaClass` through the ATL transformation rule `Cd2C`:

```
rule Cd2C{
from cd : EDOC!CompositeData
to jc : Java!JavaClass
mapsTo cd(
  name <- cd.name,
  modifier <- #regular,
  visibility <- #public,
  member <- cd.feature)
}
```

  This rule creates an instance of `Java!JavaClass` and initializes it with the same characteristics as `cd`. Thus, name is given the value of `cd.name`, `modifier` is set to `#regular`, `visibility` to `#public` and `member` is assigned with the Java equivalent of `cd.feature`.

- `ProcessComponent` is mapped to `JavaClass` through the transformation rule `Pc2C`:

```
rule Pc2C{
from pc : EDOC!ProcessComponent
to jc : Java!JavaClass
mapsTo pc(
  name <- pc.name,
```

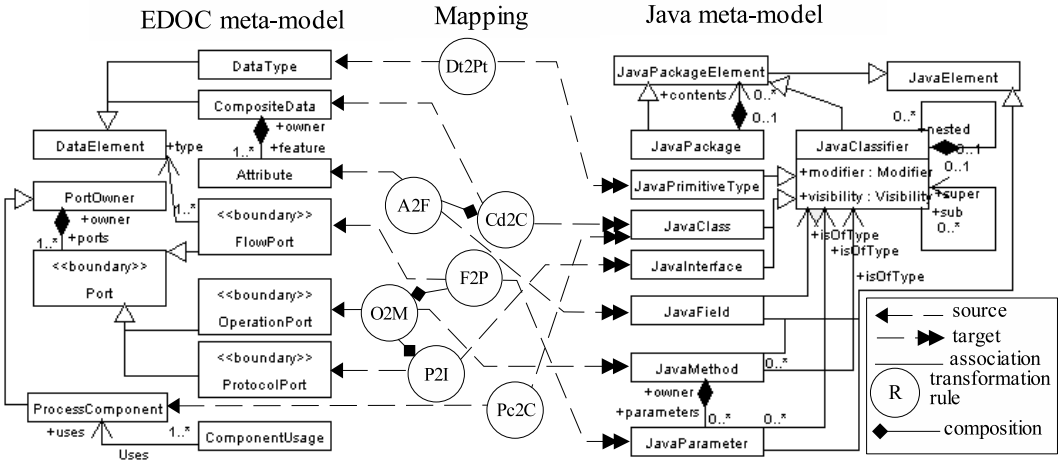

Figure 13. Mapping from EDOC to Java Meta-model

```
  visibility <- #public,
  modifier <- #regular,
  isActive <- false)
}
```

This rule creates an instance of `Java!JavaClass` and initializes it with the characteristics of `pc`. Thus, `name` is given the value of `pc.name`, `visibility` is set to `#public`, `modifier` to `#regular`, and `isActive` to `false`.

A model based on EDOC systems can be created by the utilization of UML profiles for EDOC-CCA. Then, the transformation from a PIM based on EDOC-CCA meta-model to Java meta-model can also be expressed as a transformation from UML profiles for EDOC to UML profiles for Java. However, in our approach we have privileged the direct transformation from EDOC to the Java meta-model.

5.2. From PIM (EDOC) to PSM (Web Service)

Figure 14 depicts some elements of the EDOC-CCA meta-model and the corresponding element of the WSDL meta-model. According to this figure, we have the following mappings:

- The `CCA-FlowPort` is mapped to `WSDL Part` through the rule `F2P`:

```
rule F2P_in{
from fp : EDOC!FlowPort(fp.direction=#initiates)
to p_in : WSDL!Part
mapsTo fp(
  name <- fp.type.name,
  type <- fp.type,
  owner <- [O2O.inp_m] fp.owner)
}

rule F2P_out{
from fp : EDOC!FlowPort(fp.direction=#response)
to p_out :WSDL!Part
mapsTo fp(
  name <- fp.type.name,
  type <- fp.type,
  owner <- [020.out_m] fp.owner)
}
```

In fact, this mapping is made up of two rules: `F2P_in` and `F2P_out`. `F2P_in` creates a `WSDL!Part` from the `EDOC!FlowPort` (with direction `#initiates`). `F2P_out` creates a `WSDL!Part` from the `EDOC!FlowPort` (with direction `#response`). Each `p_in` and `p_out` receives a `name` and a `type`.

- The `CCA-ProcessComponent` is mapped to `WSDL Service` through the rule `Pc2S`:

```
rule Pc2S{
from pc : EDOC!ProcessComponent
to sv : WSDL!Service
mapsTo pc(
  name <- pc.name,
  port <- [P2PType.pport] pc.owner.ports->
   select( e | e.oclIsKindOf(EDOC!ProtocolPort)))
}
```

This rule creates an instance of `WSDL!Service` and initializes it with the characteristics of `EDOC!ProcessComponent pc`. Thus, `name` is given the value of `pc.name`, `port` is assigned with the `WSDL!Port` equivalent to `EDOC!ProtocolPort`, and `location` is set to a value that must be changed before the deployment.

Analyzing figure 14, we can note that the semantics of EDOC elements are closer to that of WSDL elements than to UML elements. Thus, the mapping from EDOC to WSDL is more precise than the mapping from UML to WSDL. In this case, the mapping is more precise because it can better reproduce the target model from the source model.

In section 5.1, we obtained the Java model from the PIM using EDOC model. We can apply a similar transformation process presented in section 4.3 to generate the PSM based on JWSDP. Afterwards, the final PSM can be exported as XMI or Java code and deployment files.

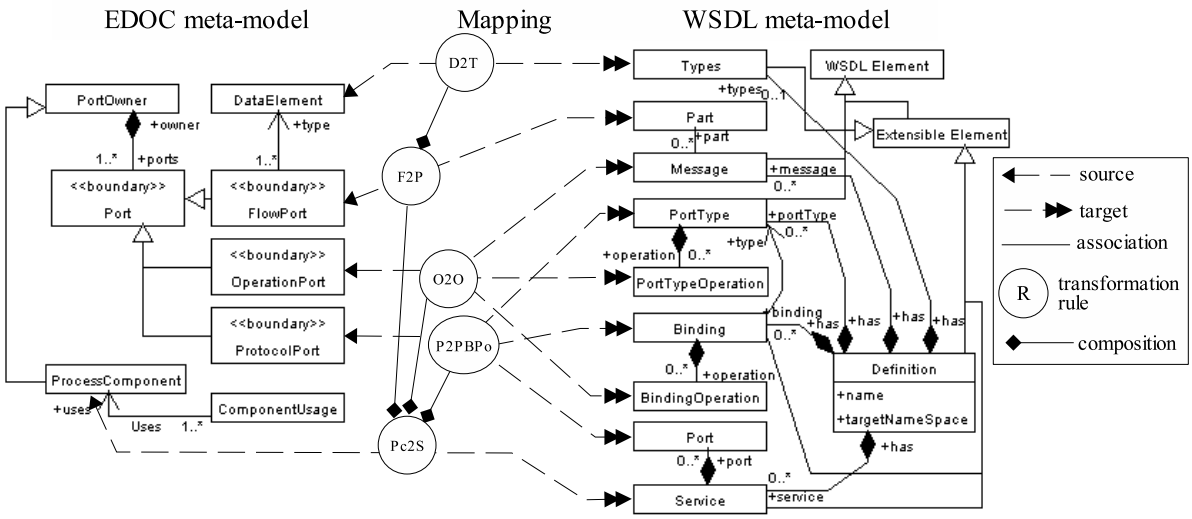

Figure 14. Mapping from EDOC to WSDL Meta-model

6. Conclusion

In the two applications of the MDA approach presented in this paper, i.e. a PIM using UML and another using EDOC, the e-business characteristics were explored without paying attention to implementation details. Afterwards, the three platform specific models (Java, Web Service and JWSDP) were generated using the ATL transformation language. One of the objectives of this experimentation was to show the importance of model transformation in the MDA approach. Furthermore, we have illustrated how well a language like ATL may be used to achieve these goals.

ATL is thus a good transformation language, but the success of an MDA approach also depends on many other factors, such as: the quality of the meta-models used to abstract and implement the system; the quality of the mapping between the source and target meta-models and the abstraction levels used in the chosen approach (e.g. business model and system model). In addition, the efficiency of transformation rules depends on the compatibility between the chosen meta-models. A transformation rule is efficient if it is capable of generating a target model element from a source model element.

EDOC provided a better representation of the functionalities and behavior patterns for a distributed system. Thus, it facilitates the generation of a more complete PSM than that obtained with UML. This was possible, because EDOC provides elements that are closer to the considered problem. Moreover, it is designed for distributed systems, while UML is more generic. Furthermore, when the PIM meta-model and the PSM meta-model present elements with equivalent structures and behaviors, the mapping is even easier.

Our experience using ATL has revealed that model transformation gives some benefits such as better formed target models. The definition of a PIM for the travel agency and the transformation rules have minimized possible errors linked with the manual creation of PSM. Moreover, this has minimized the time taken for debugging. The model transformation has also established a uniform style of development that is interesting in large projects.

MDA also has its limits and its requirements. The illustrative example has confirmed that the MDA approach cannot provide 100% of the final implementation (e.g. code, config files, deployment files). However, it is reasonable to consider that the percentage of the final implementation generated using the MDA approach is a consequence of the meta-models used. We have come to the following conclusions with our experiment:

- *Abstraction levels are necessary* - a business model should be analyzed and designed following abstraction levels. This approach efficiently assists the development of large systems;
- *The gap between meta-models can be decisive for the transformation* - the size of the gap between the source and target meta-model has a profound impact on the efforts to create transformation rules. A source model can be transformed into a target model if its respective meta-models present equivalent characteristics. If the distance between two meta-models is more significant, an intermediary meta-model may be necessary to facilitate the mapping. However, it is often impossible to transform models based on meta-models that are completely different (i.e. syntactically and semantically different). In other words, a model transformation is realizable only if the equivalent elements between the source and target meta-models can be determined;
- *UML profiles are limited* - UML profiles allow the extension of the UML meta-model. However, UML pro-

files make the creation of transformation rules difficult. In this paper, we have shown that specific meta-models based on MOF (such as Java meta-model presented in section 4.1) are more suitable to enable the creation of clear transformation rules.

In future work, we will emphasize the dynamic characteristics, i.e. the e-business' behavior and the Web Service Composition.

References

[1] M. Belaunde, J. Bézivin, and P. T. Ha. Implementing EDOC business components on top of a CCM platform. *The 7th International IEEE Enterprise Distributed Object Computing Conference (EDOC'03)*, 2003.

[2] J. Bézivin, G. Dupé, F. Jouault, G. Pitette, and J. E. Rougui. First experiments with the ATL model transformation language: Transforming XSLT into XQuery. *2nd OOPSLA Workshop on Generative Techniques in the context of Model Driven Architecture*, October 2003.

[3] J. Bézivin and N. Ploquin. Tooling the MDA framework: a new software maintenance and evolution scheme proposal. *Journal of Object-Oriented Programming (JOOP)*, December 2001.

[4] G. Caplat and J. L. Sourrouille. Model Mapping in MDA. *Workshop in Software Model Engineering (WISME2002)*, 2002.

[5] M. Carman, L. Serafini, and P. Traverso. Web Service Composition as Planning. *Workshop on Planning for Web Services*, June 2003.

[6] D. Frankel and J. Parodi. Using Model-Driven Architecture^{TM} to Develop Web Services. Technical report, IONA Technologies PLC, April 2002.

[7] A. Gerber, M. Lawley, K. Raymond, J. Steel, and A. Wood. Transformation: The Missing Link of MDA. *First International Conference on Graph Transformation (ICGT2002)*, October 2002.

[8] Java Community Process. *Java^{TM} Metadata Interface(JMI) Specification, Version 1.0*, June 2002.

[9] J. Koehler, R. Hauser, S. Kapoor, F. Y. Wu, and S. Kumaran. A model-driven transformation method. *The 7th International IEEE Enterprise Distributed Object Computing Conference (EDOC'03)*, 2003.

[10] J. Kovse and T. Härder. Generic XMI-Based UML Model Transformations. *Proc. 8th International Conference on Object-Oriented Information Systems (OOIS'02)*, pages 192–198, September 2002.

[11] J. Martin, A. Arsanjani, P. Tarr, and B. Hailpern. Web Services: Promises and Compromises. *Queue*, 1(1):48–58, 2003.

[12] Middleware Company. Model Driven Development for J2EE Utilizing a Model Driven Architecture (MDA) Approach. Technical report, The Middleware Company, June 2003. Available at www.middleware-company.com/casestudy.

[13] netBeans.org. Metadata Repository (MDR). Available at http://mdr.netbeans.org.

[14] OMG. *Model Driven Architecture (MDA)- document number ormsc/2001-07-01*, 2001.

[15] OMG. *UML Profile for Enterprise Distributed Object Computing Specification*, February 2002.

[16] S. Staab, W. van der Aalst, V. R. Benjamins, A. Sheth, J. A. Miller, C. Bussler, A. Maedche, D. Fensel, and D. Gannon. Web Services: Been There, Done That? *IEEE Intelligent Systems*, 18(1):72–85, Jan/Feb 2003.

[17] UDDI.ORG. *Universal, Description, Discovery and Integration (UDDI) Version 3.0*, July 2002. Available at http://www.uddi.org.

[18] W3C. *Simple Object Access Protocol (SOAP) 1.1*, May 2001. Available at http://www.w3.org/TR/SOAP.

[19] W3C. *Web Services Description Language (WSDL) 1.1*, March 2001. Available at http://www.w3c.org/tr/wsdl.