Generating Transformation Definition from Mapping Specification: Application to Web Service Platform

Denivaldo Lopes[1,2], Slimane Hammoudi[1], Jean Bézivin[2], and Frédéric Jouault[2,3]

[1] ESEO, France
[2] Atlas Group, INRIA and LINA, University of Nantes, France
[3] TNI-Valiosys, France
{dlopes, shammoudi}@eseo.fr
{jean.bezivin, frederic.jouault}@lina.univ-nantes.fr

Abstract. In this paper, we present in the first part our proposition for mapping specification and generation of transformation definition in the context of Model Driven Architecture (MDA). In the second part, we present the application of our proposition to Web Services platform. We propose a metamodel for mapping specification and its implementation as a plug-in for Eclipse. Once mappings are specified between two metamodels (e.g. UML and WSDL), transformation definitions are generated automatically using transformation languages such as Atlas Transformation Language (ATL). We have applied this tool to edit mappings between UML and Web Services. Then, we have used this mapping to generate ATL code to achieve transformations from UML into Web Services.

Keywords: Model Driven Architecture (MDA), Web Services, Tools for MDA.

1 Introduction

Recently, the OMG has proposed the Model Driven Architecture (MDA^{TM})[1] [1] to support the development of complex and large software systems providing an architecture with which:

- systems can evolve for meeting new requirements.
- old, current and new technologies can be harmonized.
- business logic is protected against the changes in technologies.
- legacy systems are integrated and harmonized with new systems.

In this approach, models are applied in all steps of development up to a target platform, providing source code, files of deployment and config, and so on. MDA proposes an architecture to address the complexity of software development and maintenance which has no precedents. It claims that software developers can create and maintain

[1] MDA^{TM} is a trademark of the Object Management Group (OMG).

software artifacts with little effort. However, before this becomes a mainstream reality some issues in MDA approach need solutions such as *mapping specification* and *transformation definition* [2].

In this paper, we use the term *mapping* as a synonym for correspondence between the elements of two metamodels, while *transformation* is the activity of transforming a source model into a target model in conformity with the *transformation definition*. In our approach, a transformation definition is generated from a mapping specification. The distinction between mapping specification and transformation definition is detailed in later sections.

The objective of this paper is fourfold. First, to provide a precise definition of the concepts of mapping and transformation. Second, to provide a general metamodel for mapping specification in the context of MDA. Third, to present a tool based on Eclipse enabling the editing of mappings and the generation of transformation definition from mapping specifications. Fourth, to apply our tool to Web Service Platform.

This paper is organized in the following way. Section 2 is an overview of MDA. Section 3 presents our approach for mapping specification between two metamodels in the context of MDA. Section 4 illustrates our approach applied to Web Services. Section 5 shows the implementation of our proposed metamodel for mapping through a plug-in for Eclipse, and its application to Web Services. Finally, section 6 concludes this paper and presents the future directions of our research.

2 Overview

At the beginning of this century, software engineering needs to handle software systems that become larger and more complex than before. The object-oriented and component technology seems insufficient to provide satisfactory solutions to support the development and maintenance of these systems. To adapt to this new context, software engineering has applied an old paradigm, i.e. models, but with a new aspect, i.e. Model Driven Architecture (MDA).

Some ideas around the MDA approach are not new. For example, the generation of code from a model exists from the 80's, the transformation from models into a target platform was applied some time ago to the database domain (e.g. transformation from entity-relationship to relational-tables and SQL schema). However, the standardization of an approach based on models to enable the development and maintenance of software systems is a big advance in software engineering. The change from object-oriented and component paradigm to the model paradigm was inevitable and should be irreversible. However, this does not mean the end of the former, but the introduction of models as a supplementary layer to address the development of complex and large software systems. In fact, models are the top layer and the other paradigms are the bottom layer in the MDA approach.

We cannot yet advocate that the MDA approach will resolve all problems in software system development because some issues are not well settled such as mapping, transformation, semantic distance, traceablity, and so on. However, several case studies have demonstrated that MDA is a potential solution and the future for developing software systems [3].

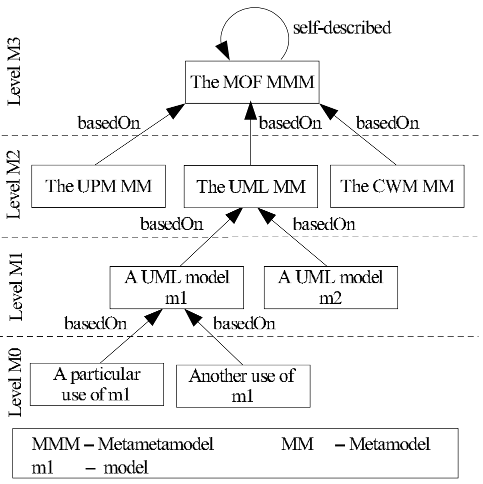


Fig. 1. Architecture with four meta-layers

2.1 The Architecture with Four Meta-layers

MDA is based on an architecture with four meta-layers [4]: metametamodel, metamodel, model and information (i.e. an implementation of its model). Figure 1 presents the idea and the relationships between different levels of models. In this approach, everything is a model or a model element, and a high level of abstraction about a problem and its solution are provided.

In level M3, a metametamodel is a well-formed specification for creating metamodels. In level M2, a metamodel is a well-formed specification for creating models. In level M1, a model is a well-formed specification for creating software artifacts. In level M0, an operational example of a model is the final representation of a software system. According to this architecture, we can envisage the existence of few metametamodels such as MOF [4] and Ecore [5], several metamodels such as UML, UEML [6] and EDOC [7], more models describing real life applications such as a travel agency, and finally infinite information such as the implementation of this travel agency model using Java. Here, it is important to pay attention to the existence of several metamodel languages, providing a Domain-Specific Language [8] or a general-purpose language (e.g. UML). In fact, the four layers are models. However, it is important to understand that each model level achieves a different goal in software development.

The development of software systems using MDA is based on the separation of concerns (e.g. business and technical concerns) which are afterwards transformed between them. So, business concerns are represented using Platform-Independent Model (PIM), and technical concerns are represented using Platform-Specific Model (PSM).

3 Mapping and Transformation

Nowadays, MDA suffers from a lack of agreement on terminology, especially concerning the concepts of mapping and transformation. In MOF QVT [2], mapping is defined as *specification of a mechanism for transforming the elements of a model conforming to a particular metamodel into elements of another model that conforms to another (possibly the same) metamodel.* In MDA distilled book [9], mapping is defined as *the application or execution of a mapping function in order to transform one model to another*, and mapping function is defined as *a collection of mapping rules that defines how a particular mapping works.* In both references and others discussed in [10], the concepts of mapping and transformation are not so clear, since these terms can refer to many different concepts. Moreover, they are usually defined without explicit distinction between them.

According to our vision, the concepts of mapping and transformation should be explicitly distinguished, and together could be involved in the same process that we denominate transformation process. In fact, in the transformation process, the mapping specification precedes the transformation definition. A mapping specification is a definition (as declarative as possible [11]) of the correspondences between metamodels (i.e. a metamodel for building a PIM and another for building a PSM). Transformation definition [2] contains a minute description to transform a model into another using a hypothetic or concrete transformation language. Hence, in our approach the transformation process of a PIM into a PSM can be structured in two stages: mapping specification and transformation definition. Finally, we define the term transformation as the manual or automatic generation of a target model from a source model, according to a transformation definition.

From a conceptual point of view, the explicit distinction between mapping specification and transformation definition remains in agreement with the MDA philosophy, i.e. the separation of concerns. Moreover, a mapping specification could be associated with different transformation definitions, where each transformation definition is based on a giving transformation definition metamodel.

Figure 2 illustrates the different concepts of MDA according to our vision where mapping specification is a mapping model, and transformation definition is a transformation model. In this figure, a mapping model is based on its metamodel, and it relates two metamodels (left and right). A transformation model is based on its transformation metamodel, and it is generated from a mapping model. A transformation engine takes a source model as input, and it executes the transformation program to transform this source model into the target model.

Several research projects have studied the mapping specification between metamodels [13] [14]. However, the ideas around mapping specification are not sufficiently developed to create efficient tools to enable automatic mappings in the context of MDA.

Nowadays, transformation languages are not yet very well explored to make choices about a standard transformation language such as desired by OMG [2]. In the next few years, the submitted propositions [15] [16] in response to QVT RFP might converge to a standard language, which will provide a new step forward in the evolution of MDA. However, wisdom tells us that one problem can be resolved using different solutions, but one solution for all problems does not exist. Thus, it is clear that this standard language

[2] In [12], transformation definition is *a set of transformation rules that together describes how a model in a source language can be transformed into a model in the target language.*

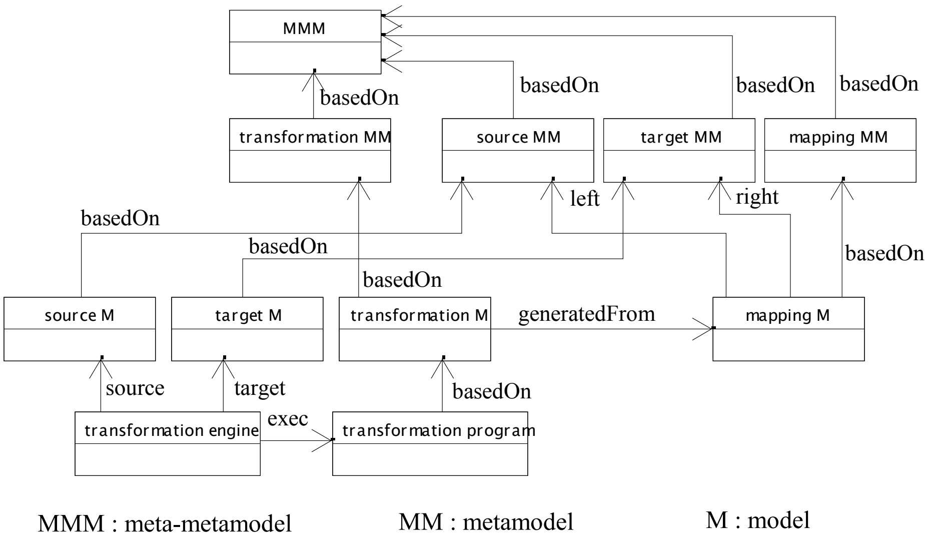


Fig. 2. Transformation process within MDA: from mapping to transformation

will not provide a sufficient solution for all types of model transformations around MDA. However, this will not be a limitation for applying MDA, because a transformation language is also a model, thus one transformation language can also be transformed into another transformation language. A priori, transformations between transformation languages seem unnecessary and unproductive. However, several examples such as Structured Query Language (SQL) (i.e. a standard query language for manipulating databases) have demonstrated that a standard is beneficial, because it establishes a unique and well-known formalism for understanding a problem and its solution. On the one hand, SQL provides a universal language for manipulating databases. On the other hand, SQL can be transformed into a proprietary language for execution into a database engine. A transformation from SQL into a proprietary language provides some benefits such as improved performance, reduction of memory-use, and so on. Making an analogy between SQL and a standard transformation language, we can expect that a standard transformation language can provide some benefits without imposing severe limitations.

Mapping and transformation have been studied for a long time ago in the database domain [11] [17]. However, they have taken another dimension with the sprouting of MDA. This does not mean that they are well-studied and ready to be applied in the MDA context. In fact, mapping specification and transformation definition are not yet an easy task. Moreover, tools to enable the automatic creation of mapping specification and automatic generation of transformation definition are still under development. Some propositions enabling the mapping specification have been based on heuristics [18] (for identifying structural and naming similarities between models) and on machine learning (for learning mappings) [19]. Other propositions enabling transformation definition have been based on graph theory [20] and compilers.

In this section, we start briefly presenting a foundation for mapping and afterwards we discuss our proposition for specifying mappings (i.e. correspondences between metamodels). This approach for mapping is based on a metamodel and implemented as a tool on Eclipse. This tool provides mapping support that is a preliminary step before the generation of a transformation definition.

3.1 Foundation for Mapping Specification

A mapping specification can be formalized as follows:

Given $M_1(s)/M_a$, $M_2(s)/M_b$, and $C_{M_a \rightarrow M_b}/M_c$, where M_1 is a model of a system s created using the metamodel M_a, M_2 is a model of the same system s created using the metamodel M_b, and $C_{M_a \rightarrow M_b}$ is the mapping between M_a and M_b created using the metamodel M_c, then a transformation can be defined as the function $Transf(M_1(s)/M_a, C_{M_a \rightarrow M_b}/M_c) \rightarrow M_2(s)/M_b$. In this section, we aim to detail $C_{M_a \rightarrow M_b}/M_c$. In general, M_a, M_b and M_c are based on the same metametamodel which simplifies the mapping specification. For now, we can define the mapping as $C_{M_a \rightarrow M_b} \supseteq \{M_a \cap M_b\}$, where $\cap$ is a binary operator that returns the elements of M_a and M_b which have equivalent structure and semantics.

We can also represent a mapping as a set. So, given:
$M_a = \{a_1, a_2, a_3, ..., a_m\}$ and $M_b = \{b_1, b_2, b_3, ..., b_n\}$
Then,
$C_{M_a \rightarrow M_b} = \{c_1, c_2, c_3, ..., c_p\}$
Where:
$c_i = \{a_k, b_j\}$
$i = \{i \in N \,|\, 1 \leq i \leq p\}$, $k = \{k \in N \,|\, 1 \leq k \leq m\}$ and $j = \{j \in N \,|\, 1 \leq j \leq n\}$.

In fact, models (i.e. in the general sense: models, metamodels and metametamodels) can be represented as sets. However, these sets are complex and heterogeneous, because their elements are classes, attributes, relationships, enumerations and datatypes. Thus, the creation of a mapping is not an easy task.

For clarity reasons, we divide elements of a metamodel into two categories: basic elements and relationships. Basic elements groups classes, attributes, enumerations and datatypes. Relationships relate classes. So, $C_{M_a \rightarrow M_b}$ must satisfy the following requirements to be a complete mapping:

1. **Basic element preservation:** each basic element of M_a must verify one of the following requirements:
   - it corresponds to an equal ($=$) basic element of M_b.
   - it corresponds to a set of basic elements of M_b that are similar ($\cong$)[3].
   - it is part of a set of basic elements from M_a that together are similar to one basic element in M_b.
2. **Relationship preservation:**
   - each relationship in M_a must verify one of the following requirements:
     - it has a corresponding relationship in M_b.
     - it has a corresponding set of relationships of M_b that are similar ($\cong$).
     - it can be implicitly preserved in M_b (i.e. through aggregating attributes or merging classes).

[3] As in [17], by similar, *"we mean that they are related but we do not express exactly how"*.

If M_b requires one basic element or relationship that can be deduced from two or more elements or relationships, respectively, from M_a, then the need for element and relationship preservation is satisfied.

If a mapping cannot satisfy the two requirements, then it is not complete, and the transformation definition is also not complete. Consequently, a target model generated from a source model using this transformation definition does not contain all the information of the source model.

The process of identifying and characterizing inter-relationships between metamodels is denoted *schema matching* [18]. In fact, *mapping* describes how two metamodels[4] are related to each other. So, *schema matching* results in a *mapping*. According to *model management algebra* [17], a *mapping* is generated using an operator called *match* which takes two metamodels as input and returns a *mapping* between them. We have adapted this operator as follow: given M_a, M_b and $C_{M_a \to M_b}/M_c$, the adapted *match operator* is formally defined as $Match'(M_a, M_b) = C_{M_a \to M_b}/M_c$.

The identification of inter-relationships between two metamodels is generally based on the metamodel structure. A metamodel structure is a consequence of relationships between its elements. These relationships have some characteristics such as *kind*. Generally, five *relationship kinds* can relate one element to another element [14]: *Association*, *Contains*, *Has-a*, *Is-a*, *Type-of*. These relationships have been used for a long time in software engineering. For example, they are common in UML: *Association* is association; *Contains* is composition; *Has-a* is aggregation; *Is-a* is inheritance; *Type-of* relates a class that is a type of an attribute. These relationship kinds can be formalized as follow:

– **Association**: $A(a, b)$ means a is associated with b.
– **Contains**: $C(c, d)$ means container c contains d.
– **Has-a**: $H(e, f)$ means e has an f.
– **Is-a**: $I(g, h)$ means g is an h.
– **Type-of**: $T(i, j)$ means i is a type of j.

In [14], the authors propose the following cross-kind-implications:

– if $T(q, r)$ and $I(r, s)$ then $T(q, s)$.
– if $I(p, q)$ and $H(q, r)$ then $H(p, r)$.
– if $I(p, q)$ and $C(q, r)$ then $C(p, r)$.
– if $C(p, q)$ and $I(q, r)$ then $C(p, r)$.
– if $H(p, q)$ and $I(q, r)$ then $H(p, r)$.

In [14], two models are considered equivalent *"if they are identical after all implied relationships are added to each of them until a fixed point is reached"*. Applying these relationship kinds and cross-kind-implications in MDA context, metamodels can be simplified and compared to find equivalences or similarities. Moreover, applying these principles and the operator $Diff'$(defined hereafter) based on the same principles presented in [17], the *semantic distance* [21] can be quantified. This operator $Diff'$ takes a metamodel M_a and a mapping $C_{M_a \to M_b}/M_c$, and returns a subset containing the elements of M_b that do not participate in the mapping. Formally,

[4] In our approach, we prefer to employ the term metamodel in the definition of the term mapping.

$Diff'(M_a, C_{M_a \to M_b}/M_c) = M_d$, where $M_d \subset M_b$. We could quantify *semantic distance* using a numeral value, but as metamodels can be considered as a set, then we prefer to quantify *semantic distance* as a sub-set of M_b. In spite of [17], we do not consider the result of $Diff'$ a sub-metamodel as expected, because we understand that this difference will return only fragments of a metamodel.

3.2 A Metamodel for Mappings

A metamodel for mapping must enable the specification of inter-relationships (i.e. correspondences) between the elements from two metamodels without modifying them. It should also provide support to handle different versions of a mapping.

Figure 3 presents a metamodel for mapping specification that meets these requirements.

In this metamodel, we consider that a mapping can be unidirectional or bidirectional. In unidirectional mapping, a metamodel is mapped into another metamodel. In bidirectional mapping, the mapping is specified in both directions. Thus, we prefer to call two metamodels in a mapping as left or right metamodel.

This metamodel for mapping presents the following elements:

- `Element` is a generalization for the other elements.
- `Historic` enables the explanation of the different choices taken for making the mapping. It has the date of the last update, a note, and the number of the last version, and a collection of `Definitions`.

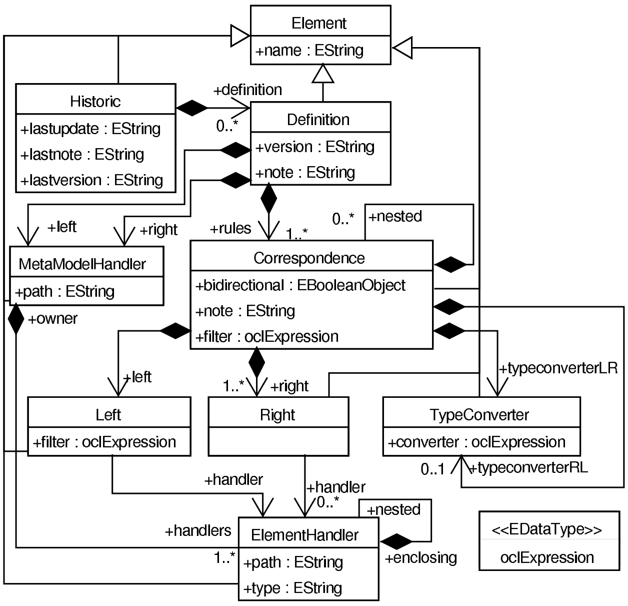


Fig. 3. Metamodel for Mapping Specification

- `Definition` is the main element and it contains all `Correspondences` between two metamodels (i.e. each correspondence has one `left` element and many `right` elements).
- `Correspondence` is used to specify the inter-relationship between two or more elements, i.e. one `left` and one or more `right` elements. The correspondence has a filter that is an OCL expression. When `bidirectional` is `false`, a mapping is unidirectional (i.e. left to right), and when it is `false` it is bidirectional (i.e. in both directions). It has two `TypeConverters` identified by `typeconverterRL` and `typeconverterLR`. `typeconverterRL` enables the conversion of the elements from a right metamodel into the elements from a left metamodel. `typeconverterLR` enables the conversion of the elements from a left metamodel into the elements from a right metamodel. Often we need to specify only the `typeconverterLR`.
- `Left` is used to identify the left element of a mapping.
- `Right` is used to identify the right elements of a mapping.
- `MetaModelHandler` is used to navigate into a metamodel. It has the information necessary for accessing a metamodel participating in a mapping. A mapping is itself a model, and it must not interfere with the two metamodels being mapped.
- `ElementHandler` enables access to the elements being mapped without changing them.
- `TypeConverter` enables the type casting between a left and a right element. If one element of a left metamodel and another element of a right metamodel are equals, then the mapping is simple and direct. However, if one element of a left metamodel and another element of a right metamodel are similar, then the mapping is complex and it is achieved using type converter, i.e. a complex expression to adapt a left element to a right element.

3.3 A Graphical Notation for Mapping

In order to simplify the mapping task, the description of a mapping specification based on our proposed metamodel for mapping should have a simplified graphical notation such as depicted in figure 4.

According to figure 4, some metamodel elements have a graphical representation. `Historic` is represented using a table. `MappingDefinition` is represented using a form that has correspondences. `Correspondence` is represented by a circle. `Left` is represented by a single arrow. `Right` is represented by a double arrow.

4 Applying MDA for Web Services

Nowadays, MDA is not sufficiently developed and experimented. In our research, we develop the MDA approach and we use Web Services as a target platform to experiment it.

Web Services have been introduced to resolve the problem of interoperability on the Internet. In fact, Web Services were created using the standards suitable for Internet. Consequently, they are more adapted to Internet than previous solutions such as

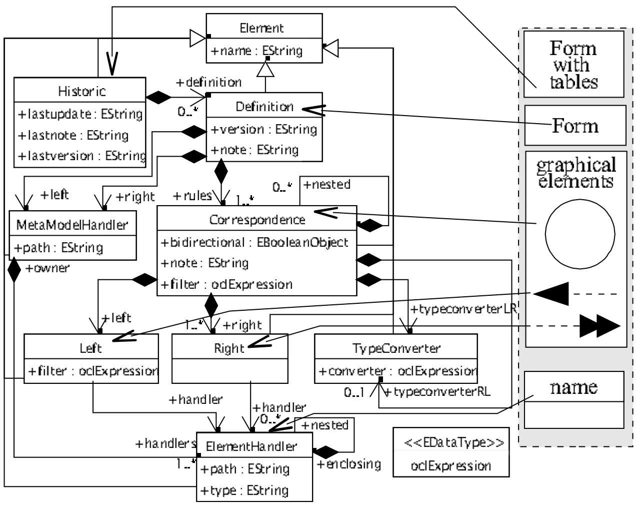


Fig. 4. Metamodel for mapping and its graphical notation

CORBA, Java RMI and EJB. However, they only provide support to the development of software systems in low level, thus the interoperability is only guaranteed in the level of implementation. MDA approach proposes an interoperability in the level of models which seems to be a promising solution. In addition, it provides mobility, i.e. a same business model can be implemented on different target platforms.

4.1 Web Services

The concept of services was introduced before Web Services. In fact, this concept has been used for a long time by OSF's Distributed Computing Environment (DCE), OMG's CORBA, Sun's Java RMI, and Microsoft's Distributed Component Object Model (DCOM) [5]. A service is an abstraction of programs, business process, and other artifacts of software defined in terms of what it does.

Service Oriented Architecture (SOA) [22] describes how a system composed of services can be built. Developing applications on SOA requires the adoption of a service-oriented design[6] which is based on the requirements determined in the strategy and business process levels.

Figure 5 shows the main SOA elements. An `AgentProvider` has `Services`. These `Services` are described through a meta-data representation, i.e. `ServiceDescription`. Afterwards, the `AgentProvider` stores information about its `Services` in a `Registry`. An `AgentRequester` searches in the `Registry` for a specific service according to a determined criterion. The `Registry` returns information about a desired service. The `AgentRequester` finds the meta-data about this service and uses it to exchange messages with the service. According to this figure, Universal Description,

[5] The actual COM+ is descendant of DCOM.

[6] Web Services is not inherently compliant to service-oriented design and to SOA.

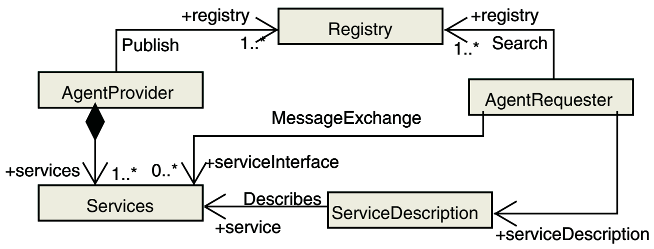


Fig. 5. Service Oriented Architecture (fragment)

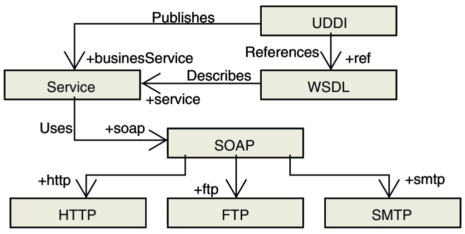


Fig. 6. Web Services (main technologies)

Discovery, and Integration (UDDI) [23] implements the `Registry`. Web Service Description Language (WSDL) [24] implements the `ServiceDescription`. `Services` use Simple Object Access Protocol (SOAP) [25] as a communication protocol, and SOAP uses HTTP or FTP or SMTP as transport protocol. Figure 6 presents the main technologies of Web Services and their relationships.

However, some issues related to Web Services still need solutions such as service composition, security and availability. Web Service composition can be static or dynamic. In a static composition, the services are determined and composed in the design time, while in the dynamic composition, the services are determined and composed at runtime. Some languages were proposed to take into account the service composition such as Business Process Execution Language for Web Services (BPEL4WS) [26].

4.2 MDA and Web Services: Mapping Specification

Web Services are the main target platform used in our experiments, and B2B applications are our privileged domain. In this paper, we present the application of our tool to map UML into Web Services, and afterwards to generate the corresponding transformation definition with Atlas Transformation Language (ATL). In order to simplify the presentation of this paper, we only show experiments with UML, WSDL [24] and BPEL4WS [26].

Figure 7 depicts a mapping specification from UML into WSDL. This representation is based on the graphical notation presented in section 3.3. For the moment, we have used this graphical notation to illustrate mappings, but we aim to introduce it in the next version of our plug-in.

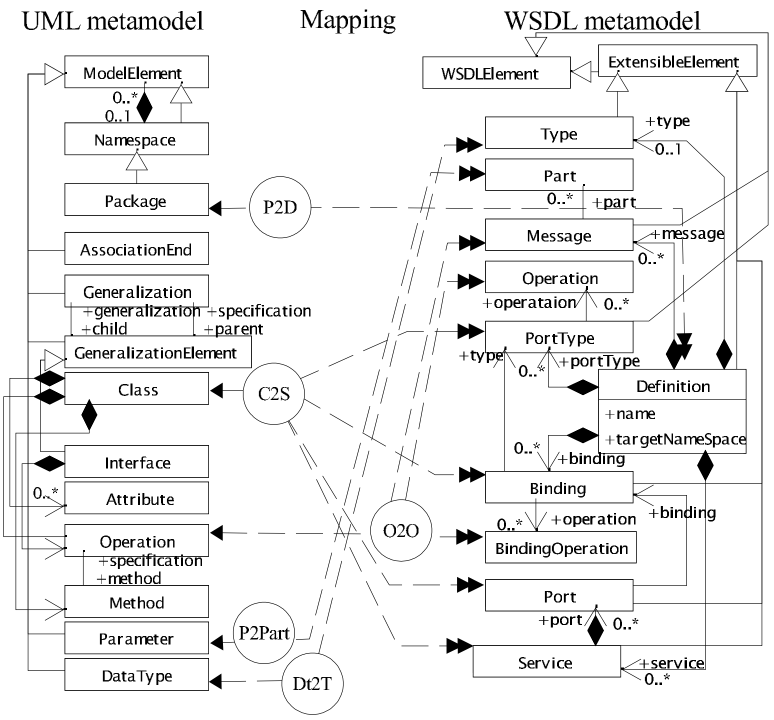


Fig. 7. A mapping from UML into WSDL (fragment)

According to figure 7, `P2D` maps `Package` into `Definition`, `C2S` maps `Class` into `Service`, `Port`, `Binding` and `PortType`[7], and so on. It is important to note that `C2S` is a mapping one-to-many, i.e. it takes one element and maps it into many elements.

5 Tool for Mapping

A tool for supporting mappings between metamodels should provide the following characteristics:

– importation of pre-existing metamodels from XMI file.
– graphical visualization of the mapping model and metamodels.
– edition of the mapping model.
– verification of conformity between mapping model and its metamodel.
– another simplified representation of a mapping model such as textual representation.
– navigation between the metamodels that are being mapped.
– semi-automatic matching.

[7] PortType was renamed to Interface in WSDL 1.2

– generation of a transformation definition from a mapping specification (i.e. mapping model).
– exportation of a mapping model using XMI file.

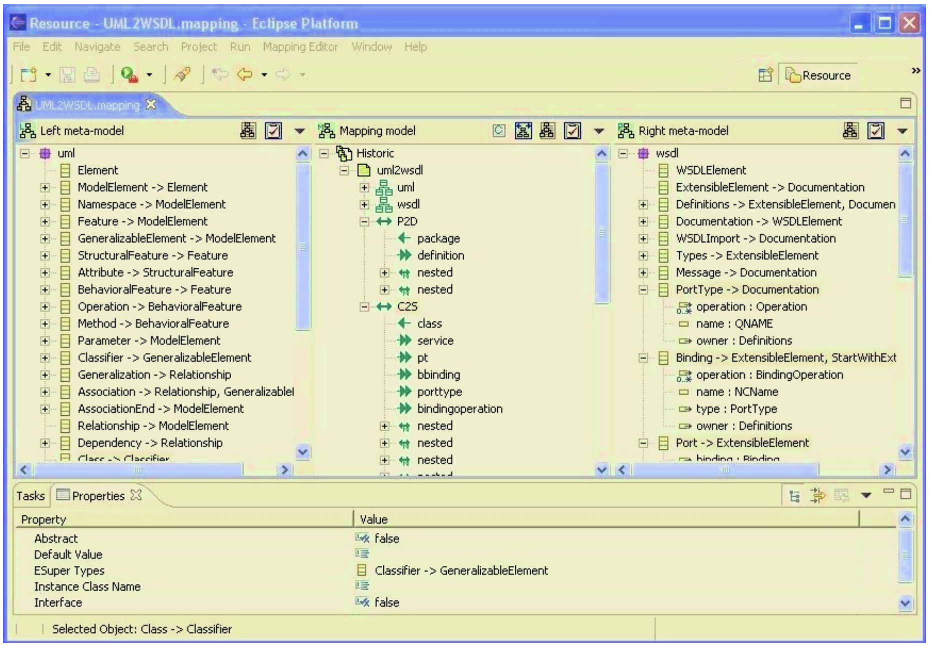
Our proposed tool supports all these characteristics, except the *semi-automatic matching* which is the next step for its improvement.

5.1 Mapping Modeling Tool (MMT)

Figure 8 shows our plug-in for Eclipse denominated Mapping Modeling Tool (MMT) that supports the mapping modeling. MMT presents a first metamodel on the left side, a mapping model in the center, and a second metamodel on the right. In this figure, the UML metamodel (fragment) is mapped into a WSDL metamodel (fragment). At the bottom, the property editor of mapping model is shown. A developer can use this property editor to set the properties of a mapping model.

Before specifying mapping using our tool, we need to create metamodels based on Ecore [5]. Some tools support the editing of a metamodel based on Ecore such as Omondo [27] or the Ecore editor provided with EMF [5]. The application of our tool using UML and WSDL metamodel can be explained in the following steps:

1. We created a project in eclipse and we imported the UML and WSDL metamodel into this project.


Resource - UML2WSDL.mapping - Eclipse Platform
File Edit Navigate Search Project Run Mapping Editor Window Help
Resource
UML2WSDL.mapping
Left meta-model
Mapping model
Right meta-model
Tasks
Properties

```
Left meta-model
uml
  Element
  ModelElement -> Element
  Namespace -> ModelElement
  Feature -> ModelElement
  GeneralizableElement -> ModelElement
  StructuralFeature -> Feature
  Attribute -> StructuralFeature
  BehavioralFeature -> Feature
  Operation -> BehavioralFeature
  Method -> BehavioralFeature
  Parameter -> ModelElement
  Classifier -> GeneralizableElement
  Generalization -> Relationship
  Association -> Relationship, Generalizablel
  AssociationEnd -> ModelElement
  Relationship -> ModelElement
  Dependency -> Relationship
```

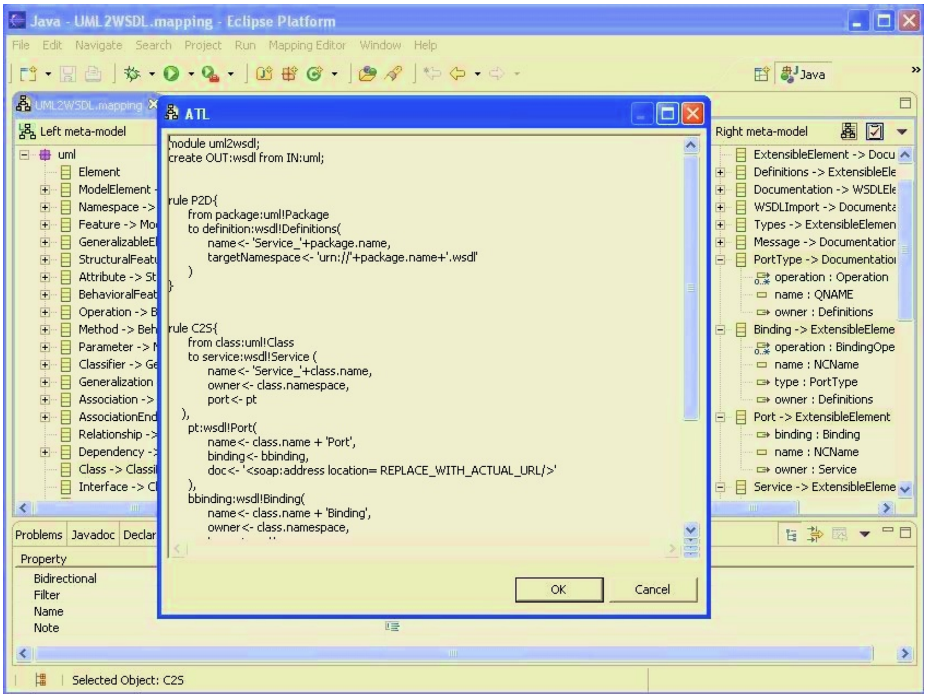
```
Mapping model
Historic
  uml2wsdl
    uml
    wsdl
    P2D
      package
      definition
      nested
      nested
    C2S
      class
      service
      pt
      bbinding
      porttype
      bindingoperation
      nested
      nested
      nested
```

```
Right meta-model
wsdl
  WSDLElement
  ExtensibleElement -> Documentation
  Definitions -> ExtensibleElement, Documen
  Documentation -> WSDLElement
  WSDLImport -> Documentation
  Types -> ExtensibleElement
  Message -> Documentation
  PortType -> Documentation
    operation : Operation
    name : QNAME
    owner : Definitions
  Binding -> ExtensibleElement, StartWithExt
    operation : BindingOperation
    name : NCName
    type : PortType
    owner : Definitions
  Port -> ExtensibleElement
```

Property	Value
Abstract	false
Default Value	
ESuper Types	Classifier -> GeneralizableElement
Instance Class Name	
Interface	false

Selected Object: Class -> Classifier

Fig. 8. Applying the tool to specify a mapping from UML into WSDL



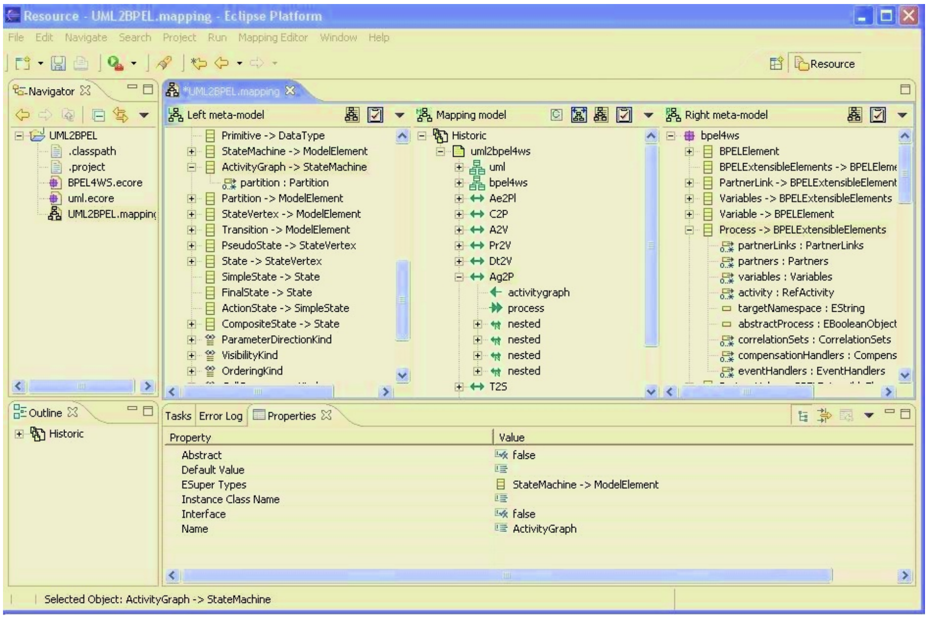
```
module uml2wsdl;
create OUT:wsdl from IN:uml;


rule P2D{
    from package:uml!Package
    to definition:wsdl!Definitions(
        name<- 'Service_'+package.name,
        targetNamespace<- 'urn://'+package.name+'.wsdl'
    )
}


rule C2S{
    from class:uml!Class
    to service:wsdl!Service (
        name<- 'Service_'+class.name,
        owner<- class.namespace,
        port<- pt
   ),
    pt:wsdl!Port(
        name<- class.name + 'Port',
        binding<- bbinding,
        doc<- '<soap:address location= REPLACE_WITH_ACTUAL_URL/>'
    ),
    bbinding:wsdl!Binding(
        name<- class.name + 'Binding',
        owner<- class.namespace,
```

Fig. 9. The generated ATL code to transform UML into WSDL

2. We used a wizard to create a mapping model. In this step, we chose the name for the mapping model, the encoding of the mapping file (e.g. Unicode and UTF-8), the files of metamodel in the format XMI.
3. The UML and WSDL metamodels are loaded from the XMI files, and the mapping model is initially created, containing the elements `Historic`, `Definition`, and `left` and `right` `MetamodelHandlers`. For each `MetamodelHandler` are also created `ElementHandlers` that are references to the elements of the corresponding metamodel.
4. We edit the mapping model. First, we define the inter-relationships between the metamodels creating `Correspondences` between their elements. Second, we create for each `Correspondence` nested `Correspondences`. Third, for each nested correspondence, we create one `Left` and one or more `Right` elements. In addition, each `Left` and `Right` element has a `ElementHandler`. If it is necessary, the `TypeConverter` is created to explicit the casting between two mapped elements.
5. The mapping model can be validated according to its metamodel, and it can be used to generate a transformation definition (e.g. using ATL language).

According to figure 8, `C2S` maps `Class` into `Service`, `Port`, `Binding` and `PortType`.

MMT can generate transformation definition from a mapping model. For the moment, we have implemented a generator for ATL[28]. The resulting code in ATL of the


Resource - UML2BPEL.mapping - Eclipse Platform
File Edit Navigate Search Project Run Mapping Editor Window Help
Resource
Navigator
*UML2BPEL.mapping
Left meta-model
Mapping model
Right meta-model
Outline
Tasks Error Log Properties

Navigator:
```
UML2BPEL
  .classpath
  .project
  BPEL4WS.ecore
  uml.ecore
  UML2BPEL.mappin
```

Left meta-model:
```
Primitive -> DataType
StateMachine -> ModelElement
ActivityGraph -> StateMachine
  partition : Partition
Partition -> ModelElement
StateVertex -> ModelElement
Transition -> ModelElement
PseudoState -> StateVertex
State -> StateVertex
SimpleState -> State
FinalState -> State
ActionState -> SimpleState
CompositeState -> State
ParameterDirectionKind
VisibilityKind
OrderingKind
```

Mapping model:
```
Historic
  uml2bpel4ws
    uml
    bpel4ws
    Ae2Pl
    C2P
    A2V
    Pr2V
    Dt2V
    Ag2P
      activitygraph
      process
      nested
      nested
      nested
      nested
    T2S
```

Right meta-model:
```
bpel4ws
  BPELElement
  BPELExtensibleElements -> BPELEleme
  PartnerLink -> BPELExtensibleElement
  Variables -> BPELExtensibleElements
  Variable -> BPELElement
  Process -> BPELExtensibleElements
    partnerLinks : PartnerLinks
    partners : Partners
    variables : Variables
    activity : RefActivity
    targetNamespace : EString
    abstractProcess : EBooleanObject
    correlationSets : CorrelationSets
    compensationHandlers : Compens
    eventHandlers : EventHandlers
```

Outline:
```
Historic
```

Property	Value
Abstract	false
Default Value	
ESuper Types	StateMachine -> ModelElement
Instance Class Name	
Interface	false
Name	ActivityGraph

Selected Object: ActivityGraph -> StateMachine

Fig. 10. Applying the tool to specify a mapping from UML into BPEL4WS

mapping between UML (fragment) and this WSDL metamodel is presented in figure 9. In this figure, a fragment of an ATL code is presented as `module uml2wsdl;...` and the rule `C2S`.

Figure 10 depicts the mapping model from UML into BPEL4WS [26] using our proposed tool. In this figure, `Ag2P` maps `ActivityGraph` into `Process`, `Dt2V` maps `DataType` into `Variable`, and so on. Since this mapping model is complete, MMT can generate the ATL code to realize transformations.

In this experiment, the ATL code was generated on the basis of the mapping model. This proposed tool left the developer free to think only at the point of the mapping between two metamodels, helping him to specify how the metamodels can be interrelated. Afterwards, it generated the code to transform models.

6 Conclusion

In this paper, we have discussed the MDA approach providing a detailed description of transformation process, distinguishing mapping and transformation. We have proposed a metamodel for mapping and a tool to support mappings. To illustrate our tool, we have specified mappings between UML as PIM and Web Services as PSM.

The *schema matching* was not yet integrated in our plug-in, because, at this stage, we are more interested in addressing the creation of mappings driven by models.

Some formalisms have been simplified and do not distinguish model, metamodel and metametamodel. Here, we have explicitly differentiated all model levels, because a model can be created using different metamodels. Moreover, the diffusion of MOF, Ecore and DSL will stimulate an increase in available metamodels.

In future research, we will develop further the graphic representation (discussed in section 3.3) and the *schema matching* in order to integrate them also into our plug-in for Eclipse.

Acknowledgments

The work described in this paper was partly financed by "Conseil Général de Maine-et-Loire", through a fellowship provided to Denivaldo Lopes.

References

1. OMG: Model Driven Architecture (MDA)- document number ormsc/2001-07-01. (2001)
2. OMG: Request for Proposal: MOF 2.0 Query/Views/Transformations RFP. (2002)
3. Middleware Company: Model Driven Development for J2EE Utilizing a Model Driven Architecture (MDA) Approach. Technical report, The Middleware Company (2003)
4. OMG: Meta Object Facility(MOF) Specification. (2002) Version 1.4.
5. Eclipse Tools Project: Eclipse Modeling Framework (EMF) version 2.0. (2004)
6. UEML.org: Unified Enterprise Modeling Language (UEML) (2003) Available at http://www.ueml.org.
7. OMG: UML Profile for Enterprise Distributed Object Computing Specification. (2002)
8. Cook, S.: Domain-Specific Modeling and Model Driven Architecture. MDA Journal (2004) 1–10
9. Mellor, S.J., Scott, K., Uhl, A., Weise, D.: MDA Distilled: Principles of Model-Driven Architecture. 1st edn. Addison-Wesley (2004)
10. Favre, J.M.: Towards a Basic Theory to Model Driven Engineering. UML 2004 - Workshop in Software Model Engineering (WISME 2004) (2004)
11. Velegrakis, Y., Miller, R.J., Popa, L.: Mapping Adaptation under Evolving Schemas. Proceedings of the 29th VLDB Conference (2003) 584–595
12. Kleppe, A., Warmer, J., Bast, W.: MDA Explained: The Model Driven Architecture: Practice and Promise. 1st edn. Addison-Wesley (2003)
13. Caplat, G., Sourrouille, J.L.: Model Mapping in MDA. Workshop in Software Model Engineering (WISME2002) (2002)
14. Pottinger, R.A., Bernstein, P.A.: Merging Models Based on Given Correspondences. Proceedings of the 29th VLDB Conference (2003) 826–873
15. DSTC, IBM, CBOP: MOF Query / Views / Transformations - Second Revised Submission. (2004) ad/2004-01-06.
16. QVT-Merge Group: Revised submission for MOF 2.0 Query/Views/Transformations RFP (ad/2002-04-10). (2004) Available at http://www.omg.org/docs/ad/04-04-01.pdf.
17. Bernstein, P.A.: Applying Model Management to Classical Meta Data Problems. Proceedings of the Conference on Innovative Data Systems Research (CIDR 2003) (2003)
18. Rahm, E., Bernstein, P.A.: A Survey of Approaches to Automatic Schema Matching. VLDB Journal **10** (2001) 334–350
19. Martin S. Lacher, G.G.: Facilitating the Exchange of Explicit Knowledge through Ontology Mappings. 14th International FLAIRS Conference (2001) 21–23

20. Agrawal, A., Levendovszky, T., Sprinkle, J., Shi, F., Karsai, G.: Generative Programming via Graph Transformation in the Model-Driven Architecture. OOPSLA 2002 Workshop on Generative Techniques in the Context of Model Driven Architecture (2002)
21. Bézivin, J., Hammoudi, S., Lopes, D., Jouault, F.: Applying MDA Approach for Web Service Platform. 8th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2004) (2004) 58–70
22. W3C: Web Services Architecture (WSA). (2004)
23. UDDI.ORG: Universal, Description, Discovery and Integration (UDDI) Version 3.0. (2002)
24. W3C: Web Services Description Language (WSDL) 1.1. (2001)
25. W3C: Simple Object Access Protocol (SOAP) 1.1. (2001)
26. Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., Weerawarana, S.: Business Process Execution Language for Web Services (BPEL4WS) version 1.1. (2003)
27. Omondo: Omondo Eclipse UML. (2004) Available at http://www.omondo.com.
28. Bézivin, J., Dupé, G., Jouault, F., Pitette, G., Rougui, J.E.: 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 (2003)