Journal of Universal Computer Science, vol. 18, no. 17 (2012), 2364-2382
submitted: 16/1/12, accepted: 29/8/12, appeared: 1/9/12 © J.UCS

Towards Model-Driven Engineering Support for Service Evolution

Juan M. Vara
(Kybele Research Group, Rey Juan Carlos University
Madrid, Spain
juanmanuel.vara@urjc.es)

Vasilios Andrikopoulos
(Institute of Architecture of Application Systems, University of Stuttgart
Stuttgart, Germany
vasilios.andrikopoulos@iaas.uni-stuttgart.de)

Michael P. Papazoglou
(European Research Institute in Service Science (ERISS)
Tilburg University, Tilburg, The Netherlands
m.p.papazoglou@uvt.nl)

Esperanza Marcos
(Kybele Research Group, Rey Juan Carlos University
Madrid, Spain
esperanza.marcos@urjc.es)

Abstract: In the field of Service-Oriented Architecture (SOA) evolution is a key issue given the non-trivial nature of updating widely distributed and heterogeneous systems. With this in mind, in this work we used some of the technologies developed in the context of the Eclipse Modeling Framework (EMF) to provide a proof of concept of the possible synergy between Model-Driven Engineering (MDE) and Service Orientation. In particular, we present a DSL toolkit for modeling the structural part of Abstract Service Descriptions (ASDs) and the reasoning mechanism that assesses whether two versions of a service are compatible with respect to its consumers.

Keywords: Service Evolution, Model-Driven Engineering, Compatibility, Type Theory
Categories: D.2.1, D.2.2, D.2.6, D.3.1, H.1.1, F.4.3

1 Introduction

Model Driven Engineering (MDE) is a recent trend in software engineering; its main proposal is to focus on models rather than on computer programs [Selic 03]. The basic assumption in MDE is to consider models as first class entities, just as classes are the basic construction block in object-oriented programming, or software components are the basic unit in component-based software engineering. Indeed, MDE is a natural step in the historical tendency of software engineering towards raising the abstraction level at which software is designed and developed. Although models have always been considered in software development, they have been traditionally used simply as

documentation, and in the best case, they have served to generate a reduced skeleton of the final code. With the advent of MDE the landscape has changed, at least for MDE practitioners that have shifted their focus from coding to modelling.

During the last years, MDE started to have a direct influence in other research fields. Under the light of the premise that *everything is a model* [Bézivin 04], practitioners from other software development areas have discovered that they are able to express their SE problems in MDE terms and take advantage of MDE techniques to solve them – or at least simplify them, by leveraging the level of automation in the development process. The scope varies widely from more generic domains, like Web Engineering [Koch et al. 08] or Service Orientation [Bell 08] to more specific ones, like DB schema matching [Bernstein 03] or domotics [Jimenez et al. 09]. In order to express their SE problems in terms of MDE, they need metamodelling frameworks, model transformation languages and the like. Following this line, our work focuses on providing methods and modelling languages to assist Service-Oriented Development. In particular, it aims at showing how MDE tools and techniques are applied to provide support for a *service evolution framework*.

Service evolution is the disciplined approach of managing service changes and is defined as *the continuous process of development of a service through a series of consistent and unambiguous changes* [Andrikopoulos 10]. The evolution of a service is expressed through the creation and decommissioning of different *service versions* during its lifetime. These versions must be aligned with each other in such a way as to allow a service developer to track the various modifications introduced over time and their effects on the original service. To control service development, a developer needs to know why a change was made, what its implications are, and whether the resulting service version is *compatible* with existing consumers. A service version is compatible if it does not render its consumers inoperable (i.e., it does not *break* them in the sense that consumers are still able to use the same type of data they used as inputs and get back the say type of data they got as outputs).

Several approaches for controlling service evolution like [Becker et al. 08; Brown and Ellis 04] depend on a set of empirical guidelines in order to enforce service compatibility. These guidelines prescribe the type of changes that can occur to the description of a service interface (usually a WSDL document). This dependence on guidelines however is limited in expressivity and portability to other technologies since it relies on the specifics of a particular version of WSDL. For this purpose, [Andrikopoulos 10] proposes a theoretical framework that allows the formal definition of the conditions under which the evolution of service interfaces respects service compatibility. The proposed framework depends on formal models for the representation of service interfaces that draw from a common metamodel. This property makes it ideal for applying MDE techniques and serving as a proof of concept for the synergy between MDE and Service Orientation.

The rest of this article is structured as follows: Section 2 briefly presents the formal framework for compatible service evolution developed in [Andrikopoulos 10] in order to provide the theoretical background for this work. Section 3 demonstrates the synergy between MDE and SOA by discussing how the theoretical framework can be implemented by means of MDE techniques and tools. Section 4 validates our work through the use of a case study. Finally, Section 0 discusses related work and Section 6 concludes and presents our intentions with respect to future work.

2 Compatible Service Evolution Framework

[Andrikopoulos 10] presents a rigorous formal framework based on type safety criteria and algorithms which controls and delimits the evolution of services. The framework extends and applies theories and methods of controlling evolution from object-oriented programming languages like subtyping and co- and contra-variance of input and output [Meyer 97]. Based on these principles a reasoning mechanism is presented that allows deciding when a change to the service interface leads to a compatible version of the service for the consumers.

More specifically, the framework is based on a technology-agnostic notation for the representation of service interfaces in the form of *Abstract Service Descriptions (ASDs)* as introduced in [Andrikopoulos et al. 08]. Each ASD represents a particular version of a service and can be defined as the set S of all its versioned records $S = \{s_i^j\}, i = 1,\ldots,N, j \in V$, where V is the set containing the version identifiers *vid* for all records *s*. *S* therefore contains one particular version for each of its constituent records. Each ASD respects a particular metamodel, expressed in UML class diagram notation in 0, which divides ASD records in three layers: a structural, a behavioural and a non-functional one. For the purposes of this discussion, we focus on the structural layer.

The structural layer of the ASD, depicted in the lower part of 0, contains the method signatures and their message parameters required for the interaction of the clients with the service. In particular, it includes the following concepts:

- **Information Type** is a wrapper for the XML Schema complex and simple data types that are used as parts of the message exchange. For representing simple data types, each Information Type contains the valueType and valueRange properties. valueRange expresses the allowed range of values for each Information Type (or N/A if one is not defined). valueType belongs to the DataType property domain which contains the usual simple data types from XML Schema like int, double, string, etc. The document value is used for complex data types. Since complex types may contain both simple and other complex types, the actual content of the complex types is expressed through the (optional) reflexive association relationship with other Information Types. No information is explicitly stored about the particular structure of the type (e.g. the sequence of the nested elements inside a complex type).
- **Message** corresponds to the WSDL message and message part elements. It is the container of the message payload and for that purpose, its property role draws values from the MessageRole property domain. MessageRole contains the three basic roles that a message can play in an interaction with a consumer: input, output and fault (that is, an exception-like output). A Message must contain at least one Information Type for “storing" the message content.
- **Operation** represents the basic interaction point of the clients with the service in the form of a discrete functionality to be performed. It contains one or more Messages, as defined by the semantics of its pattern. The MessagePattern property do main in Fig. 4.1 contains the four interaction patterns with a service (one-way, notification, request-response and solicit-response) as defined in WSDL 1.* Each interaction pattern binds the number and properties of the

Messages it is related to. request-response for example would mean that the Operation would be connected to (at least) two Messages, one with property input and one with output (and optionally one with property fault). More powerful interaction patterns, or even customly defined ones, as provided by WSDL 2.0 and discussed in the Adjuncts section of the specification can also be used here, as long as their semantics are reflected accordingly in the relationship of the Operation with its Messages. A 'robust-in-only' message pattern for example would have signified that there will be exactly one Message of with property input and so on.

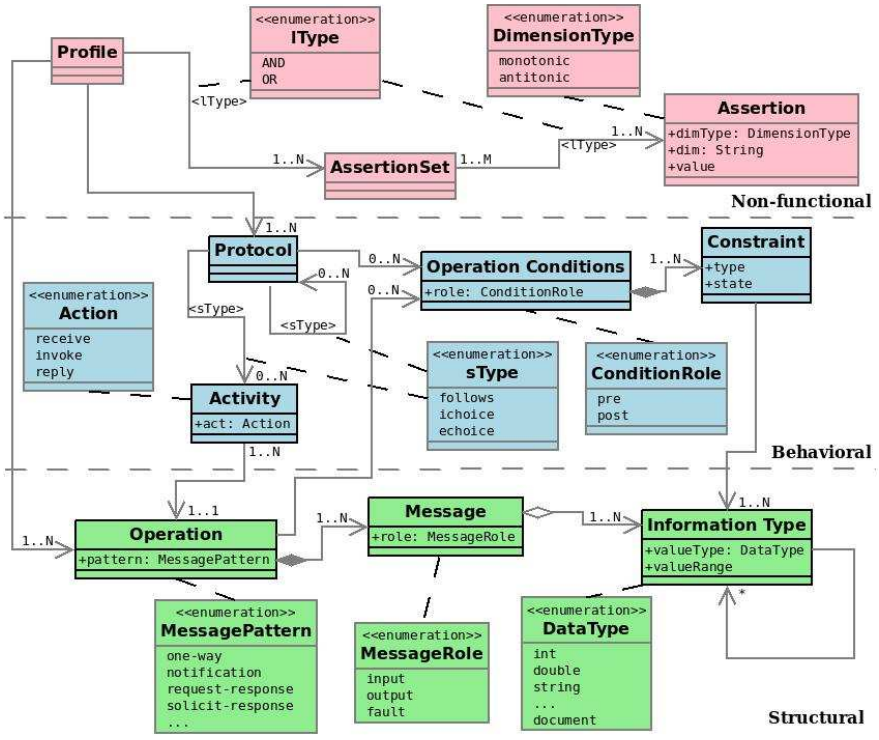

Figure 1: The ASD metamodel

The theoretical aspect of the ASD notation uses a formal specification of the ASD notation based on type theory [Cardelli 98]. A structural ASD consists of *elements* – informational constructs representing the building blocks of the service – and their relationships – expressing the structural dependencies of elements. Elements and their relationships are formally defined as:

Definition 1. An element e is a tuple $e := (name : string, (pr_{i\geq 1} : property)^*)$. A relationship $r(e_s, e_t)$ between elements e_s and e_t is a tuple $r(e_s, e_t) := (name : string, name_s : string, name_t : string, rel : relation, mul : multiplicity)$ where:

- $name, name_s, name_t$ are unique string identifiers of elements e, e_s, e_t respectively, e.g., *RequestMessage*.
- $(pr_{i\geq 1} : property)^*$ is a set of zero or more *properties*, drawing from a *property domain*. The `messagePattern` to be used for an operation is an example of a property domain, containing properties like `One-way`, `Request-Response`, etc. The domains for each property are depicted as enumerations in 0.
- *rel* is the type of type of relationship between the elements (*a,c,s* – that is, aggregation, composition or association, respectively, with the semantics defined in [Andrikopoulos et al. 08]).
- *mul* is the *multiplicity* of the relationship, defined as $mul := [\min, \max]$, where $\min, \max \in \mathrm{N}$ (the set of natural numbers) is the minimum and maximum respectively multiplicities allowed for each member of the relationship, as denoted in 0.

To illustrate the correspondences between the concepts of ASD metamodel and the formal concepts of *elements* and *relationships*, Listing 2 shows the elements and relationships used by the structural layer of the ASD metamodel.

Listing 1. Formal specification of the ASD metamodel

```
1. Operations: o := (name : string, messagePattern)
              where messagePattern∈{one-way, notification,
              request-response, solicit-response}.
2. Messages: m := (name : string, role)
              where role∈{input, output, fault}.
3. InformationTypes: it := (name : string, valueType, valueRange)
              where valueType  {document, int, float, double,
              string,…} and valueRange ∈ {(min, max), N/A}.
-------------------------------------------------------------------
Relationship types:
1. Operation-Message: r(o,m) := (name_o, name_m, c, mul)
              where mul = [1, 1..*].
2. Message-InformationType: r(m, it) := (name_m, name_it, a, mul)
              where mul = [1, 1..*].
3. InformationType-InformationType: r(it_i, it_j) := (name_i, name_j ,
              s, mul) where mul = [0, 1..*].
```

For instance, an Operation is an element that owns just one property (messagePattern), while an Operation-Message relationship is containment relationship between one Operation element and one or more Message objects.

Next, note that elements e and relationships $r(e_s, e_t)$ are *records* in the type theoretical sense which allows us to define a *subtyping* relation on them:

Definition 2. An element $e := (name, pr_1, \ldots, pr_k)$ is a subtype of element $e' := (name', pr'_1, \ldots, pr'_m)$, and we write $e \leq e'$, iff $name \equiv name' \wedge m \leq k \wedge pr_i \leq pr'_i, 1 \leq i \leq m$, that is, they have the same name identifier, and e' has less properties than e, but the ones it has are more generic (super-types) of the respective properties of e.

A relationship $r(e_s, e_t)$ is a subtype of relationship $r(e'_s, e'_t)$, and we write $r(e_s, e_t) \leq r(e'_s, e'_t)$, iff $e_s \leq e'_s \wedge e_t \leq e'_t \wedge rel = rel' \wedge mul \subseteq mul'$, that is, the elements participating in the (new) relationship are super-types of the original ones and the multiplicity domain of the relationship is a super-set of the respective one in the old relationship.

Using the subtype relation and a segmentation of an ASD S into two proper subsets S_{pro} and S_{req}, denoting output- and input-specific elements and relationships respectively we can easily check for the compatibility of two ASDs as follows:

Definition 3. Two service versions S and S' are called compatible, and we write $S <_c S'$, iff $\begin{cases} \forall s' \in S'_{req} \exists s \in S_{req} : s \leq s' \text{ (contra - variance of input)} \\ \forall s \in S_{pro} \exists s' \in S'_{pro} : s' \leq s \quad \text{(co - variance of output)} \end{cases}$

Note that this notion of compatibility is not symmetric: the fact that S is compatible with S' does not imply that S' is compatible with S. It just implies that the data types of their inputs and outputs allows replacing S by S' without breaking the consumers. Note also that we are not considering here the functionality of S and S'. One might assume that they provide the same functionality or develop behavioural checks to assess it. Furthermore, from their definition, ASD models assume a functional, input/output-oriented view of services in line with the object-oriented paradigm that they evolved out of [Meyer 97]. However, different models of services, such as for example the AS^2 model for data-intensive services [Ma et al. 09], do not adopt this functional view. As such, the theoretical tools discussed in this model are not directly applicable and appropriate, equivalent mechanisms have to be developed (if possible) in order to allow for the remaining of our proposed approach to be used in conjunction with them.

The fact that Definition 3 can be easily translated into a *compatibility checking function*, combined with the fact that ASDs can be expressed as models conforming to a common metamodel, allow for a direct application of MDE techniques for implementing the compatible service evolution framework as we discuss in the following section.

3 Modeling and Comparing Abstract Service Descriptions

In order to support the theoretical framework discussed above we developed a DSL for ASDs and a basic toolkit to use it. In particular:

- We encoded the structural aspect of the ASD metamodel discussed above.
- We developed both a tree-like editor and a diagrammer for models that conform to the ASD metamodel.
- We implemented a basic comparison between ASDs to assess whether they are compatible according to the principles defined in the last section.

The development process we followed to accomplish these tasks is shown in 0. It is adapted from the process proposed in [Vara 09] for the development of new DSL toolkits. Each step in the development process is represented with a rounded rectangle while the software artefacts produced are represented with ovals. Besides, how each product is used in subsequent steps of the development process is also depicted. In

addition, the main technical solution used for each task is represented by means of its corresponding logo.

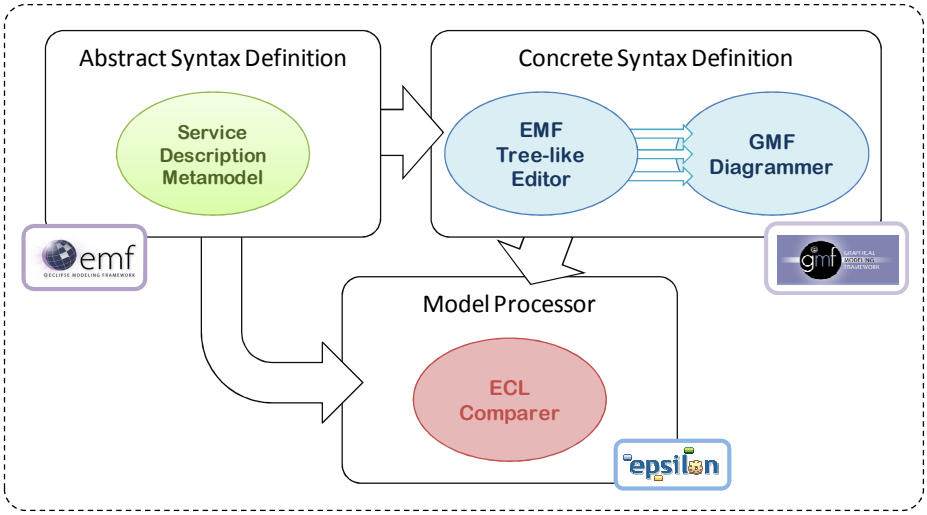

Figure 2: SRMod development process

First step is the definition of the abstract syntax of the DSL. To this end, the metamodel of the DSL is defined as an Ecore metamodel using the facilities provided by the Eclipse Modelling Framework (EMF) [Budinsky et al. 08], like the Ecore tools and the Ecore diagrammer.

Next task is the definition of a concrete syntax for the DSL. EMF and GMF (Graphical Modelling Framework) [Tikhomirov and Shatali 08] are used for this purpose to generate a couple of model editors, a basic tree-like editor and a diagrammer. Due to the GMF architecture, the latter is based on the former, which helps with subsequent steps of the process.

Finally, once the DSL has been defined it is time to address the model processing tasks for which it was devised. In this case, the aim was to compare ASDs. For this purpose we developed a model comparer using the Epsilon Comparing Language [Kolovos 09]. The following sections present the key points for each step of the process.

3.1 Abstract Syntax Definition

The first task to address when deploying a DSL is the specification of its metamodel which collects the abstract syntax of the language. It describes the vocabulary of the concepts provided by the language and how they may be combined to create new models. Listing 2 contains the elements and relationships used by the structural layer of the ASD metamodel.

Listing 2. Formal specification of the ASD metamodel

```
1. Operations: o := (name : string, messagePattern)
              where messagePattern∈{one-way, notification,
              request-response, solicit-response}.
2. Messages: m := (name : string, role)
              where role∈{input, output, fault}.
3. InformationTypes: it := (name : string, valueType, valueRange)
              where valueType  {document, int, float, double,
              string,…} and valueRange ∈ {(min, max), N/A}.
-------------------------------------------------------------------
Relationship types:
1. Operation-Message: r(o,m) := (name_o, name_m, c, mul)
              where mul = [1, 1..*].
2. Message-InformationType: r(m, it) := (name_m, name_it, a, mul)
              where mul = [1, 1..*].
3. InformationType-InformationType: r(it_i, it_j) := (name_i, name_j ,
              s, mul) where mul = [0, 1..*].
```

To implement this metamodel we have used Emfatic[1], a language that allows for representing EMF Ecore models in textual form. The use of Emfatic allows introducing some `@gmf` annotations during the definition of the metamodel that will serve to drive the look & feel of the GMF diagrammer. Next, the Emfatic textual specification is injected into an Ecore metamodel that includes the `@gmf` annotations which are later used to drive the generation of the diagrammer. 0 shows two partial views of this metamodel.

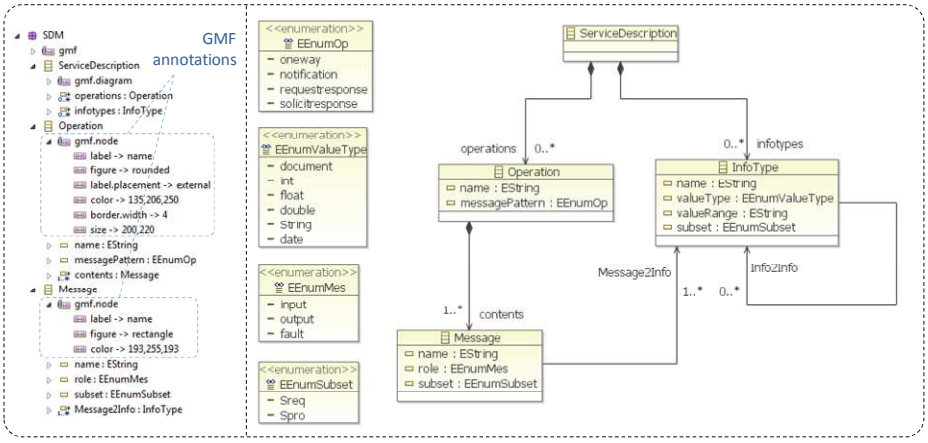

Figure 3: The ASD metamodel as an Ecore metamodel

The left-hand side shows the default EMF tree-like editor. Notice how the `@gmf` annotations have been mapped to Ecore annotations. The right-hand side of the picture shows the metamodel depicted with the Ecore diagrammer that supports the representation of Ecore metamodels as class diagrams.

[1] http://www.alphaworks.ibm.com/tech/emfatic

3.2 Concrete Syntax Definition

After defining the metamodel of the DSL, the next step is to provide it with a concrete syntax. Broadly speaking, defining the concrete syntax of a DSL consists of associating a notation to each concept and relationship in the metamodel. We have focused on visual notations since visual representation of models has been typically identified as one of the technological foundations supporting MDE [Bézivin 04]. Moreover, we are currently working to import ASD models directly from WSDL files so we could use WSDL as the textual notation.

To address this task, EMF and GMF capabilities are used to generate a tree-like editor with basic capabilities, and a diagrammer. We believe that any DSL toolkit has to bundle a diagrammer for each DSL supported, since they are useful to provide with a first glance of any given model. However, the effort dedicated to their development should be considered very carefully. In this context, the generative nature of GMF fits perfectly: it generates an efficient – though not perfect, diagrammer in reasonable time and with minimum effort. Given the above, the process to provide with a (graphical) concrete syntax for the ASD DSL in EMF is summarized as follows:

- From the Ecore (meta)model, EMF provides runtime support for graphically editing, manipulating, reading, and serializing data based on the given (meta)model. Thus, conforming models can be created using a simple tree-like editor.
- Since we rely on extended EMF tree-like editors as the most convenient way of handling low-level models, the next step is to customize the basic EMF tree-like editor. Examples of this process can be found in [Vara 09]. It is worth mentioning that GMF relies extensively on EMF-generated code. The way model elements are displayed on GMF diagrammers, the icons used to identify them, and even the labels that show their names are directly taken from EMF generated code. Thus, the improvements made over the EMF tree-like editor are automatically transferred to the GMF diagrammer.
- Finally, the EuGENia[2] and the KybeleGMFgen[3] plug-in are used to leverage the level of automation of the model-driven development process for diagrammers implemented by GMF. This automated process consumes three types of inputs: the Ecore meta-model, the code that implements the EMF tree-like editor and a set of Epsilon Object Language (EOL) files. Since a predefined set of annotations is not enough to obtain exactly the required look & feel for the diagrammer, the EOL files collect the different design decisions that we want to project in the diagrammer. From these inputs a chain of model transformations generate a new set of intermediate models and finally the working code that implements the diagrammer.

3.3 Comparing ASDs

In the following we present a first implementation of the comparison between two service versions to assess whether they are compatible. According to [Andrikopoulos 10], we can summarize ASD version compatibility by: *two service versions*

[2] http://www.eclipse.org/gmt/epsilon/doc/eugenia/
[3] http://kybelegmfgen.wordpress.com/

(represented as ASDs) are compatible if one of them is composed by the same elements and relationships (or a subtype of them) that compose the other. In other words, two service versions are compatible if (sub)typing relations hold between their elements and relationships. The sufficient conditions for subtyping between the components of an ASD are summarized in Listing 3.

Listing 3. Conditions for subtyping

1. $o \leq o' \Leftrightarrow name = name' \wedge messagePattern = messagePattern'$ (we only accept equality).
2. $m \leq m' \Leftrightarrow name = name' \wedge role = role'$ (as above).
3. $it \leq it' \Leftrightarrow name = name' \wedge valueType \leq valueType' \wedge valueRange \subseteq valueRange'$

   where $int \leq float \leq double \leq string$ (everything else is not valid) and the value ranges are compared as subsets (if applicable).
4. $r(o, m) \leq r'(o,m) \Leftrightarrow o \leq o' \wedge m \leq m' \wedge mul \subseteq mul'$.
5. $r(m, it) \leq r'(m, it) \Leftrightarrow m \leq m' \wedge it \leq it' \wedge mul \subseteq mul'$.
6. $r(it_i, it_j) \leq r'(it_i, it_j) \Leftrightarrow it_i \leq it'_i \wedge it_j \leq it'_j \wedge mul \subseteq mul'$.
7. For all relationships, it holds that

   $\varnothing \leq r(s, t) \Leftrightarrow [0, 0] \subseteq mul$,

   or equivalently, if the relationship has a multiplicity of [0, 1..*]).

To implement the comparison, we translate the conditions of Listing 2 into a operative Epsilon program using the Epsilon Comparison Language, a hybrid rule-based language built atop the Epsilon platform, which enables developers to implement comparison algorithms at a high level of abstraction [Kolovos 09]. The next section will show the result of running the comparison between different versions of a service in practice.

The result of this implementation process was the SRMod prototype, which is publicly available at http://srmod.wordpress.com.

4 Case Study

To validate the different software artifacts produced in this work we have used the Automotive Purchase Order Processing Scenario which is being developed and used as one of the validation scenarios in the S-Cube Network of Excellence[4]. The scenario is based on the Supply Chain Operations Reference (SCOR) model that provides guidelines for the implementation of Supply Chains. This scenario is an example of how to implement activities of SCOR Level 3 using SOA, based on the processes of a company belonging to the automobile industry called Automobile Incorporation (aka AutoInc). AutoInc contains various business units, for example, Sales, Logistics, Manufacturing, etc., and cooperates with other partners such as suppliers, banks,

[4] http://www.s-cube-network.eu/

carriers, etc. In the following we assume that the various activities are implemented as services.

In particular, we used the SRMod prototype to model two different versions of one service (`Receive Purchase Order`). The service contains one `Operation` that handles two different `Messages` (`POMessage` and `m2`) and uses a set of data types like `PODocument`. Each version differs from the other in the signature of the operations provided. The first version of the considered service is shown in Listing 4.

Listing 4. Formal specification of the Receive Purchase Order Service (V1)

```
o1 = (processOrder, request - response)
m1 = (POMessage, input)
m2 = (m2, output) //for unnamed output message
it1 = (PODocument, document, N/A)
it2 = (OrderInfo, string, N/A)
it3 = (DeliveryInfo, string, N/A)
it4 = (it4, string, N/A) //for unnamed output message
r(o1, m1) = (processOrder, POMessage, c, [1, 1])
r(o1, m2) = (processOrder, m2, c, [1, 1])
r(m1, it1) = (POMessage, PODocument, a, [1, 1])
r(m2, it4) = (m2, it4, a, [1, 1])
r(it1, it2) = (PODocument, OrderInfo, s, [1, 1])
r(it1, it3) = (PODocument, DeliveryInfo, s, [0, 1])
```

The graphical representation of the service description is depicted in 0, along with some screen captures showing the properties sheet for each type of object.

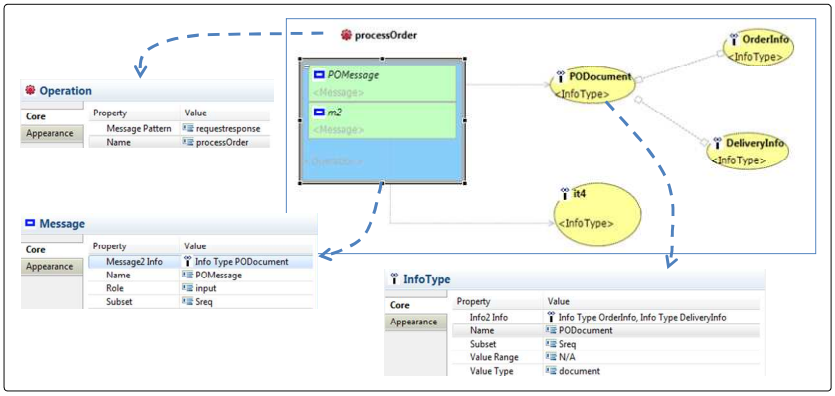

Figure 4: Graphical representation of Receive Purchase Order Service (V1)

The second version of the `Receive Purchase Order` service adds a new `InfoType` object to model the date of the purchase order (`OrderDate`) and a new relationship, which relates a given purchase order with its date of expedition. Again, the new version of the service is formally described in Listing 5 (new elements are in bold).

Listing 5. Formal specification of the Receive Purchase Order Service (V2)

```
o1 = (processOrder, request – response)
m1 = (POMessage, input)
m2 = (m2, output) //for unnamed output message
it1 = (PODocument, document, N/A)
it2 = (OrderInfo, string, N/A)
it3 = (DeliveryInfo, string, N/A)
it4 = (it4, string, N/A) //for unnamed output message
it5 = (OrderDate, date, N/A)
r(o1, m1) = (processOrder, POMessage, c, [1, 1])
r(o1, m2) = (processOrder, m2, c, [1, 1])
r(m1, it1) = (POMessage, PODocument, a, [1, 1])
r(m2, it4) = (m2, it4, a, [1, 1])
r(it1, it2) = (PODocument, OrderInfo, s, [1, 1])
r(it1, it3) = (PODocument, DeliveryInfo, s, [0, 1])
r(it1, it5) = (PODocument, OrderDate, s, [0, 1])
```

Given the above, when the comparison between the two service versions is performed, then the results are displayed in a structured way in the Eclipse console. We are currently working to transform them into a new model that will be later processed by means of MDE techniques.

The results are split into two big groups of model elements that inform respectively of the compatibility between objects (instances of the metaclasses collected in the metamodel) and relationships (instances of the meta-associations collected in the metamodel). For each model element found on one version (element or relationship), we check if the other service version contains an element whose properties fulfil the requirements for replacing the former without affecting service compatibility.

The lower part of 0 shows that all the partial comparisons yield a positive result when V1 is used as the old version of the service, i.e., V1 can be replaced by V2. Indeed, despite that the `PODocument` data type uses an additional data type (`OrderDate`) in V2, we can still replace V1 by V2 without affecting compatibility since we can look at `V1.PODcoument` as a subtype of `V2.PODocument`. On the contrary, if V2 is used as the service to be replaced (by V1), individual comparisons yield a negative result since the subtyping relations are not fulfilled: we cannot replace the `V2.PODocument` data type with the `V1.PODocument` data type, as shown in Fig. 5.

In summary, with the SRMod prototype we have replaced the tedious and error prone task of parsing two WSDL files to look for differences, expressed them in a manner suitable for computing the formal algorithms described in Section 2 and finally displayed the results in a user-friendly way. At the same time, we provided with a graphical representation of the ASD that helps on its understanding and provides with a quick overview of each service description.

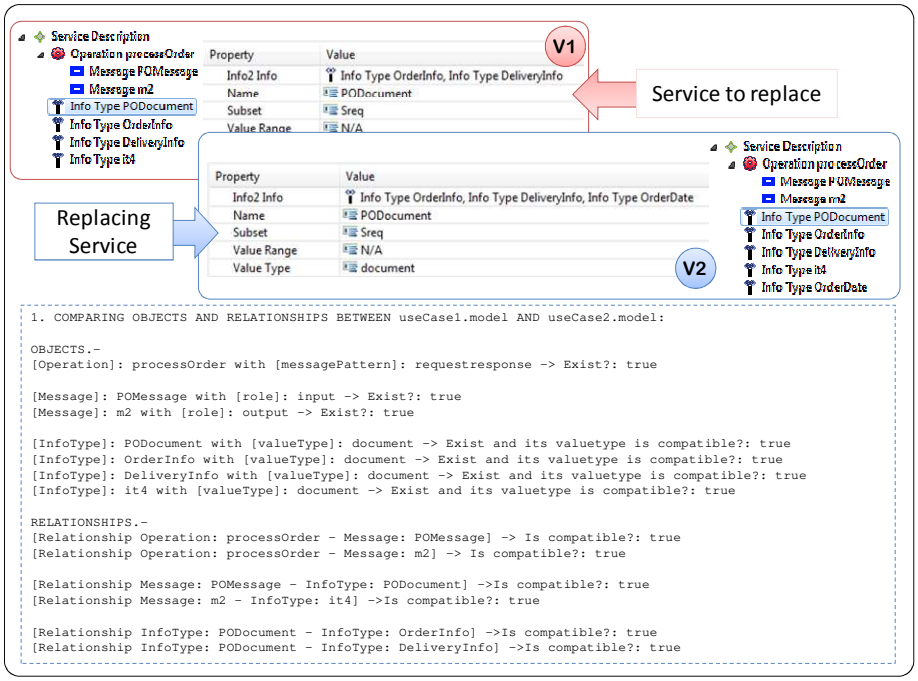

Figure 5: Excerpt from the comparison results: V1 used as former version

5 Related Work

Evolution in software systems has been traditionally considered as either a part, or a synonym of *software maintenance* and defined by the empirical laws that drive and govern the evolution of software systems [Lehman 96]. However, attempting to apply the conventional maintenance procedure (halt operation, edit source and re-execute) in service-oriented environments is not sensible [Bennet and Raljich 07]. The difficulty of identifying which software artifacts constitute the system itself is non-trivial, especially in the context of large service networks. In addition, the lack of ownership and access to the actual source code (if any) of third-party services (due to the SOA principles of encapsulation and loose coupling) does not allow the application of many of the maintenance techniques like refactoring or impact analysis. This has given rise to different approaches in service evolution.

On the one end of the spectrum there are approaches on service evolution that do not consider whether the changes to a service version break the consumers of the service, preferring to remain as neutral as possible e.g. [Juric et al. 09; Leitner et al. 08]. On the other end, there are approaches that aim to enforce non-breaking changes of services to the extent that versioning of the service description can be simply subsumed under one version, the active (i.e., deployed and running) one. A special case of this idea is proposed by [Endrei et al. 06] where there are two versions active

at all times: the current one and the old version which will be deprecated within a given time period.

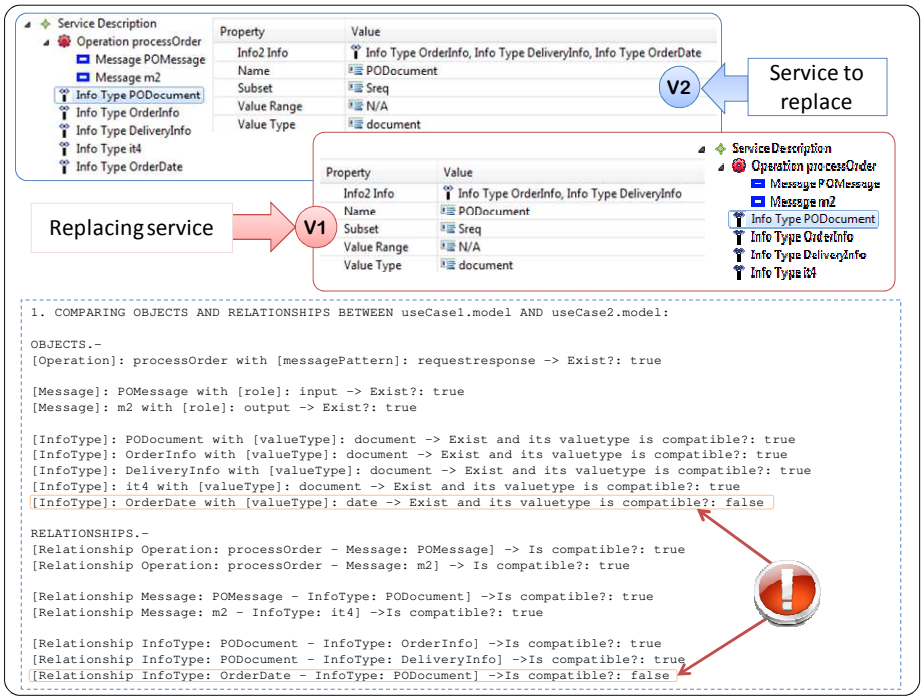

Figure 6: Excerpt from the comparison results: V2 used as former version

The majority of the approaches in the literature, e.g., [Becker et al. 08; Brown and Ellis 04; Fang et al. 07; Weinreich et al. 07] are located somewhere between these two ends. In principle, they propose a common compatibility-oriented strategy for service evolution: maintain multiple active service versions for major releases, but cut maintenance costs by grouping all minor releases under the latest one. However, such works lack the theoretical framework to support the empirical guidelines they depend on. By contrast, we aim at providing a holistic theoretical and technological framework to support service evolution. Such framework operates on a formal basis, focused on enforcing non-breaking changes to service versions.

With respect to the discovering of model differences, it is worth mentioning the works from Bernstein [Bernstein 03] around schema matching. In the MDE field there are several works focused on identifying differences between UML models, such as [Ohst et al. 03] or [Xing and Stroulia 05]. The work from [Sriplakich et al. 06] raises the level of abstraction since it allows discovering differences between models conforming to any given metamodel. Finally, in the context of EMF, EMF Compare[5] is probably the most accepted tool for detecting changes between different models.

[5] http://wiki.eclipse.org/index.php/EMF_Compare

Indeed it serves as the basis for more elaborate proposals on metamodel evolution and model co-evolution, like [Cichetti et al. 09]. Nevertheless, all these tools are more oriented towards automatically discovering equivalences and differences. Since we needed to implement our own algorithm to establish such relationships we have opted for using the Epsilon Comparison Language because it is specifically intended for this type of tasks.

As regards the convergence of MDE and SOA, some authors argue that such an effort, and in combination with better techniques for documenting and improving business processes, are the way to realize the promise of rapid, accurate development of software that serves, rather than dictates, software users' goals [Watson 08]. This kind of alignment between high level business specifications and lower level SOA implementations is a crucial aspect in the field of Service-Oriented Development. Besides, a number of methodological proposals have appeared trying to combine the strengths of MDE and SOA [Andrikopoulos 10; Arsanjani et al. 08; Bell 08] by providing models, methods and techniques to deal with the development of service-oriented systems. Most of them however do so at a very high level of abstraction, based around business process modeling. Our approach is applicable to the level of service version, making it more accessible in practice for service developers.

6 Conclusion and future work

The prototype presented in this work is complete in the sense that the current version of SRMod is the first release of a service Evolution framework: it supports the modeling of the structural part of Abstract Service Descriptions (ASDs) and provides with a reasoning mechanism that assesses whether two versions of a service are compatible with respect to its consumers.

However, we would like to note that the development of this prototype has served as some kind of *position prototype*: it has served to show the potential advantages that MDE technologies can bring to the deployment of SOA methodological proposals. As a consequence, as long as we were going forward in the development of SRMod, we found a number of issues that will be addressed by means of other MDE techniques. For instance, now that we are able to model ASDs, we are able to address the automatic mapping of WSDL files into ASDs (and vice-versa).

Therefore, in the following we enumerate some of the future challenges that we plan to address in the development of our Service Evolution framework. Note that most of them are already ongoing work:

- First, we are already working on the improvement of the produced software artifacts. In particular, we are extending the DSL to deal not only with the structural layer of the ASD metamodel, but also with the non-functional one (top layer of Fig. 4).
- Besides, we are building a set of extractors and injectors to bridge the ASD DSL with languages for Web service description such as WSDL. This will allow us to import and export directly from and to WSDL documents. To that end we are using the TCS (Textual Concrete Syntax) language [Jouault et al. 06]. With TCS, it is possible to parse (text-to-model) and pretty-print (model-to-text) DSL sentences. To that end, TCS provides with a DSL for the specification of the

correspondence between the metamodel and its textual representation. From that, an ANTLR grammar together with a parser for this grammar is generated. Such parser (also known as injector) takes as input a textual program of the DSL and generates a model conforming to the DSL metamodel.

- Furthermore, we are extending the use of MDE technologies in handling the result of the comparison process. More specifically, version comparison results will be expressed in a *weaving model* that relates the two source models. To that end, we use the ATLAS Model Weaver (AMW). The AMW workbench provides a set of standard facilities for the management of weaving models and metamodels [Didonet Del Fabro and Valduriez 08]. This way, each element of the weaving model would inform of the level of compatibility between two given elements of the Service versions compared, together with a short description of the issues related to the compatibility assessment.
- Unfortunately, the theoretical framework that serves as methodological basis for the proposal, does not deal with the name space and schemas issues since it abstract the comparison of data types with the concept of "Information Type". To address this issue, we will take advantage from existing MDE technologies. In particular, the EMF metamodeling framework that constitutes the technological basis of this proposal bundles a complete set of facilities for modeling XML Schemas and dealing with theirs instances (conforming XML documents). We are already using them to ease the injection of WSDL files into ASD models since WSDL files are nothing but XML files conforming to a given XML Schema. Hence, we will use them also to improve the comparison of XSD data types.

Finally, as the next step in demonstrating the suitability of MDE techniques in a SOA setting we plan to apply a similar approach to the one described in this paper to existing work on *service contracts* [Andrikopoulos et al. 09]. A service contract is an intermediary construct interposed between service providers and consumers, expressed also in ASD form, which can be used to represent a technical Service Level Agreement (SLA) between them. The use of contracts allows for greater flexibility in evolving both interacting parties (i.e., providers and consumers) in a compatible manner. Furthermore, the work of [Andrikopoulos et al. 09] shows that even the contract itself can evolve under certain conditions. The fact that the authors (re-)use the ASD notation and the subtyping relation as discussed in Section 2 provide an ideal setting for an extension of our current work.

Acknowledgments

This research has been carried out in the framework of the MODEL-CAOS (TIN2008-03582) and the MASAI (TIN-2011-22617) projects, financed by the Spanish Ministry of Science and Innovation, and has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement 215483 (S-Cube).

The authors would also like to thank David Granada at Kybele Research group for his help with the SRMod prototype.

References

[Andrikopoulos 10] Andrikopoulos, V.: "A Theory and Model for the Evolution of Software Services"; Tilburg: Tilburg University Press. Available: http://arno.uvt.nl/show.dgi?fid=107815 (2012).

[Andrikopoulos et al. 08] Andrikopoulos, V., Benbernou, S., and Papazoglou, M.: "Managing the evolution of service specifications"; Proc. of the International Conference on Advanced Information Systems Engineering (CAiSE'08). Springer-Verlag (2008), 359-374.

[Andrikopoulos et al. 09] Andrikopoulos, V., Benbernou, S., and Papazoglou, M.: "Evolving services from a contractual perspective"; Proc. of the International Conference on Advanced Information Systems Engineering (CAiSE'09), Springer-Verlag (2009), 290-304.

[Arsanjani 08] Arsanjani, A., Ghosh, S., Allam, A., Abdollah, T., Ganapathy, S., and Holley, K.: "SOMA: A method for developing service-oriented solutions"; IBM Systems Journal, 47, 3 (2008), 377-396.

[Becker et al. 08] Becker, K., Lopes, A., Milojicic, D., Pruyne, J., and Singhal, S.: "Automatically determining compatibility of evolving services"; Proc. of the IEEE International Conference on Web Services (ICWS 2008), 161-168.

[Bell 08] Bell, M.: "Service-Oriented Modeling (SOA): Service Analysis, Design, and Architecture"; Wiley (2008).

[Benett and Raljich 07] Bennett, K.H. and Rajlich, V.T.: "Software maintenance and evolution: a roadmap"; Proc. International Conference on The Future of Software Engineering (ICSE), ACM (2007), 73-87.

[Bernstein 03] Bernstein, P A. "Applying Model Management to Classical Meta Data Problems"; Proc. First Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA (2003).

[Bézivin 04] Bézivin, J.: "In search of a Basic Principle for Model Driven Engineering"; Novatica/Upgrade, V(2) (2004), 21-24.

[Bézivin 05] Bézivin, J.: "On the Unification Power of Models. Software and Systems Modeling"; 4, 2 (2005), 171-188.

[Brown and Ellis 04] Brown, K. and Ellis, M.: "Best practices for web services versioning"; IBM (2004). Available: http://www.ibm.com/developerworks/webservices/library/ws-version/

[Budinksky et al. 08] Budinsky, F., Merks, E., Steinberg, D.: "Eclipse Modeling Framework 2.0 (2nd Edition)"; Addison-Wesley Professional (2008).

[Cardelli 98] Cardelli, L.: "A semantics of multiple inheritance"; Information and Computation, 76, 2-3 (1998), 138-164.

[Cicchetti et al. 09] Cicchetti, A., Ruscio, D., and Pierantonio, A.: "Managing Dependent Changes in Coupled Evolution"; Proc. 2nd international Conference on theory and Practice of Model Transformations, Zurich, Switzerland. Springer-Verlag, Berlin (2009), 35-51.

[De Castro et al. 09] De Castro, V., Marcos, E. and Wieringa, R.: "Towards a Service-oriented MDA-Based Approach to the Alignment of Business Processes with it Systems: From the Business Model to a WS Composition Model"; Int. Journal on Cooperative Systems, 18, 2 (2009), 225-260.

[Didonet Del Fabro and Valduriez 2008] Didonet Del Fabro, M., Valduriez, P.: "Towards the efficient development of model transformations using model weaving and matching transformations"; Software Systems Modeling, 8, 3 (2008), 305-324.

[Endrei et al. 06] Endrei, M., Gaon, M., Graham, J. Hogg, K. and Mulholland, N.: "Moving forward with Web services backward compatibility"; (2006). [Online]. Available: http://www.ibm.com/developerworks/java/library/ws-soa-backcomp/index.html?ca=drs.

[Fang et al. 07] Fang, R., Lam, L., Fong, L., Frank, D., Vignola, C., Chen, Y. and Du, N.A.: "Version-aware approach for web service directory"; Proc. IEEE International Conference on Web Services (ICWS'07) (2007), 406-413.

[Jimenez et al. 09] Jimenez, M., Rosique, F., Sanchez, P., Alvarez, B., Iborra, A.: "Habitation: A Domain-Specific Language for Home Automation"; IEEE Software, 26, 4 (2009), 30-38.

[Jouault et al. 06] Jouault, F., Bezivin, J. and Kurtev, I.: "TCS: a DSL for the Specification of Textual Concrete Syntaxes in Model Engineering"; Proc. GPCE'06 Fifth international conference on Generative programming and Component Engineering, Portland, Oregon, USA (2006), 249-254.

[Juric et al. 09] Juric, M.B., Sasa, A. Brumen, B. and Rozman, I.: "WSDL and UDDI extensions for version support in web services"; Journal of Systems and Software, 82, 8 (2009), 1326-1343.

[Koch et al. 08] Koch, N, Meliá, S. Moreno, N. Pelechano, V. Sanchez, F. & Vara, J.M.: "Model-Driven Web Engineering"; Upgrade Journal, IX, 2 (2008), 40-46.

[Kolovos 09] Kolovos, D.S.: "Establishing Correspondences between Models with the Epsilon Comparison Language"; Proc. 5th European Conference on Model Driven Architecture - Foundations and Applications (ECMDA-FA '09). Springer-Verlag (2009), 146-157.

[Lehman 96] Lehman, M.M.: "Laws of software evolution revisited"; Proc. 5th European Workshop on Software Process Technology (EWSPT), Springer-Verlag (1996), 108-124.

[Leitner et al. 08] Leitner, P., Michlmayr, A., Rosenberg, F. and Dustdar, S.: "End-to-End versioning support for web services"; Proc. IEEE International Conference on Services Computing (2008), 59-66.

[Ma et al. 09] Ma, H. Schewe, K.-D., Thalheim, K.-D., Wang, Q: "A Theory of Data-Intensive Software Services"; Service Oriented Computing and Its Applications, 3, 4 (2009), 263-283.

[Meyer 97] Meyer, B.: "Object-Oriented Software Construction, 2nd ed"; Upper Saddle River, NJ, USA: Prentice Hall PTR (1997).

[OMG 01] Object Management Group, "MDA Guide Version 1.0"; (2001), OMG Document - omg/2003-05-01.

[Ohst et al. 03] Ohst, D., Welle, M., Kelter, U.: "Differences between versions of UML diagrams". SIGSOFT Softw. Eng. Notes, 28, 5 (2003), 227-236.

[Paige et al. 09] Paige, R.F., Kolovos, D.S., Rose, L.M., Drivalos, N. and Polack, F.: "The Design of a Conceptual Framework and Technical Infrastructure for Model Management Language Engineering"; Proc. IEEE International Conference on Engineering of Complex Computer Systems (IECCS) (2009), 162-171.

[Selic 03] Selic, B.: "The pragmatics of Model-Driven development"; IEEE Software, 20, 5 (2003), 19-25.

[Sriplakich et al. 06] Sriplakich, P., Blanc, X., Gervais, M.P.: "Supporting collaborative development in an open MDA environment"; Proc. 22nd IEEE International Conference on Software Maintenance (ICSM'06), IEEE Computer Society (2006), 244-253.

[Tikhomirov and Shatali, 08] Tikhomirov, A., and Shatali, A.: "Introduction to the Graphical Modeling Framework";. Tutorial at the EclipseCON 2008. Santa Clara, California (2008).

[Vara 09] Vara, J.M.: "M2DAT: a technical solution for Model-Driven Development of Web Information Systems"; PhD Thesis. University Rey Juan Carlos (2009). Available: http://www.kybele.etsii.urjc.es/members/jmvara/Thesis/.

[Watson 08] Watson, A.: "A Brief History of MDA"; Upgrade, IX, 2 (2008), 7-11

[Weinreich et al. 07] Weinreich, R., Ziebermayr, T. and Draheim, D.: "A versioning model for enterprise services". Proc. 21st International Conference on Advanced Information Networking and Applications Workshops (AINAW'07), (2007), 570-575.

[Xing and Stroulia 05] Xing, Z., Stroulia, E.: "UMLDiff: an algorithm for object-oriented design differencing". Proc. 20th IEEE/ACM international Conference on Automated software engineering (ASE '05), New York, NY, USA, ACM (2005), 54-65.