PAPER

A Technique for Defining Metamodel Translations

Iván GARCÍA-MAGARIÑO[†a)], *Member and* **Rubén FUENTES-FERNÁNDEZ**[†b)], *Nonmember*

SUMMARY Model-Driven Engineering and Domain-Specific Modeling Languages are encouraging an increased used of metamodels for the definition of languages and tools. Although the Meta Object Facility language is the standard for metamodeling, there are alternative metamodeling languages that are aimed at satisfying specific requirements. In this context, sharing information throughout different domains and tools requires not only being able to translate models between modeling languages defined with the same metamodeling language, but also between different metamodeling languages. This paper addresses this latter need describing a general technique to define transformations that perform this translation. In this work, two case studies illustrate the application of this process.

key words: *Model Transformation, Metamodel Translation, Model-Driven Engineering*

1. Introduction

Metamodeling is [1] the study and development of the theories, rules, constraints, and models used for modeling in a given domain. It is becoming increasingly popular for the definition of Domain-Specific Modeling Languages (DSML) [2]–[4] and it is standard procedure in Model-Driven Engineering (MDE) [5]. Among the advantages of metamodeling are its aim of defining graphical modeling languages, the availability of frameworks to build support tools, and the possibility of manipulating the models through automated transformations, at least in MDE.

In short, metamodeling languages (MMLs) describe metamodels that characterize modeling languages (MLs). The Object Management Group (OMG), which is the main standardization organism for metamodeling, has defined a metadata architecture [6] to explain the relations between the previous elements in the definition of languages. This architecture considers four layers of abstraction:

- The Meta-Metamodel layer (M3). This is the layer of the MMLs such as the Meta Object Facility [6] language.
- The Metamodel layer (M2). M2 is the layer where the metamodels of MLs are defined using MMLs from M3. For instance, the definition of the Unified-Modeling Language (UML) [7], [8] metamodel in M2 uses MOF from M3.
- The Model layer (M1). It contains specific models defined from the metamodels in M2. In this layer specific UML diagrams for a given problem could appear.
- The User Objects layer (M0). M0 defines the concrete objects of a problem as instances of models in M1.

The current standard MML is the Meta Object Facility (MOF) [6] language defined by the OMG. Nevertheless, there is a variety of MMLs targeted to satisfy specific requirements. Some of them emerge from MOF as Essential MOF (EMOF) and Complete MOF (CMOF) [6], which are the subsets of MOF that organize its description. For DSMLs, the INRIA (Institut National de Recherche en Informatique et en Automatique, France) has developed the KM3 [9] language for metamodeling. GOPRR (Graph Object Property Relationship Role) [10] is the MML used by the MetaEdit+ tool. The ECore [11] language of the Eclipse Modeling Framework (EMF) [12] is probably the most widely used MML for implementation due to its Eclipse support. It is almost aligned with EMOF [12]. Other platforms like MS/DSL [13] or Coral [14] also include their own metamodeling languages.

This variety of MMLs makes the exchange of information difficult between solutions that adopt different proposals to define their modeling languages (MLs). Transformations between MMLs have been proposed for this purpose, in the same way that they are already used to share information between MLs based on the same MML. Existing literature contains examples of ad-hoc metamodel translations, like the one for the translation between MOF and ECore in [15]. However, these works do not explicitly define a general technique for the development of metamodel translations, since they are more like guidelines or catalogues of correspondences between abstractions for specific cases.

There are two facts that make the definition of general metamodel translations difficult. First, the meaning and level of abstraction of the different MMLs vary. For instance, both GOPRR [10] and ECore [11] contain elements to connect entities. GOPRR uses relationships and ECore uses references. Nevertheless, GOPRR relationships can contain attributes in the body or in the ends of the relationships, whereas ECore references do not have attributes. Even in cases where the languages are more similar such as MOF and ECore [15], the definition of transformations has to deal with non-aligned languages that do not have the same expressive power. The second problem is that model transformation languages rely on unique specific MMLs. That

Manuscript received October 9, 2008.

[†]The authors are with the Dept. of Software Engineering and Artificial Intelligence, Facultad de Informática, Universidad Complutense de Madrid, Spain.

a) E-mail: ivan_gmg@fdi.ucm.es

b) E-mail: ruben@fdi.ucm.es

DOI: 10.1587/transinf.E92.D.2043

is, for a given transformation language, a transformation needs its source and target MLs to be defined in the same MML. For instance, the ATLAS Transformation Language (ATL) [16] relies on ECore. However, the meta-metamodels of these MMLs are not usually defined in other MMLs and therefore there is no convenient transformation between these MMLs. Thus, model-transformation languages cannot usually be used for defining metamodel translations. As an exception, KM3 [9] is defined with ECore, and an ATL transformation can translate between KM3 and ECore MMLs.

This paper presents a general technique for defining metamodel translations. This technique relies on the categorization of the elements or element-patterns of metamodeling languages. This classification is based on the fact that most MMLs are graph-based, so they share concepts such as node or edge. The technique includes several steps for: the identification of correspondences between elements of MMLs, the propagation of their attributes and links, the disambiguation of conflicts in the translation, and the implementation of transformation rules. A conflict in the translation appears when an element in the target metamodel could be the result of the translation of different elements of the source metamodel. For instance, an *EClass* of ECore is required to represent both *Objects* and *Relationships* from GOPRR, since both *Objects* and *Relationships* have properties but in ECore only *EClasses* can have properties. Depending on the case, the technique defines its rules with a different approach according to the classification in [17]. If there are no conflicts, the technique adopts a declarative approach that only considers the correspondences between elements, attributes and links; if conflicts appear, additional package-like structures must be added to the target and a structure-driven approach is better suited to this task. In both cases, the experimentation of this technique is carried out on general-purpose transformation languages. Existing model transformation languages are not suitable for metamodel translations since they usually need both the source and target models to be defined in the same MML.

The structure of the remainder of the paper is as follows. The next section provides a background on model transformations. Section 3 presents the technique developed in this research for translating metamodels. Then, Sect. 4 gives a detailed description of two case studies of metamodel translation defined using this technique. This work is contextualized and compared with related research in Sect. 5. Finally, conclusions and future work are discussed in Sect. 6.

2. Background

Metamodel translations are a particularization of model transformations in which models are metamodels and the semantic of the source remains in the target model. Model transformations can be classified into the following approaches [17]:

Approach 1: *Direct-manipulation approaches*. The models have some kind of internal representation and an API to manipulate it. Generally, these transformations are managed with imperative languages such as Java.

Approach 2: *Declarative approaches*. These are based on mathematical relations. A relation indicates the type of source and target elements and uses constraints for its definition. Satisfying these constraints is necessary for the rules to be applied. An example with the Mercury variant of Prolog appears in the work of Gerber et al. [18]. For the purpose of this discussion, graph approaches (see [17]) are considered a particular type of declarative approach.

Approach 3: *Structure-driven*. In model transformation literature [16], this approach is also called *imperative*. Transformations in this category have two phases, the first one creates the hierarchical structure of the target model, and the second one fills in the attributes and references in the target model. An example of this approach is the framework provided by OptimalJ [19].

Approach 4: *Hybrid approaches*. They combine techniques from the other approaches. For instance, ATL [16] combines declarative and imperative approaches.

The technique presented in this paper uses a hybrid approach, declarative and structure-driven, to define the metamodel translations. The choice depends on the features of the translation. If it does not need to organize elements in the target with some package-like structure, the declarative approach is preferred given its flexibility and declarative expression of transformations. Otherwise, the declarative approach is complemented with a structure-driven one because it facilitates the insertion of new elements and its rearrangement in the package-like structure.

In order to implement the technique, the chosen transformation language must be able to satisfy the previous requirements. Given the hybrid (both declarative and structure-driven) nature of the technique, we have chosen standard model transformation languages such as ATL [16], Query/View/Transformation (QVT) (or some of its variants) [20], or XSL Transformations (XSLT) [21]. They allow both approaches to be implemented although biased to a declarative expression.

The previous transformation languages can be classified into two groups. Some of them define transformations that rely on the availability of the metamodels for the involved MLs. These metamodels allow the compliance of the source and target models to be validated. The metamodels need to be defined with a particular metamodeling language, like ECore in the case of ATL or MOF for QVT. Hence, this kind of language is not usually applicable to the problem of translation between MMLs, since the MMLs of the source and target models may be different. A second group of languages defines transformations between models of different modeling languages without an explicit definition of their metamodels. These transformations have the advantage of being very flexible since there are few constraints on the

MLs that they can process. Nevertheless, they are less robust than the previous transformations since they cannot validate their source and target models unless this check is implemented with their own transformations. These languages are known as general-purpose transformation languages. The most widely used is XSLT [21], which is originally intended to transform general XML documents. An example of the use of XSLT for metamodel translations is the work of Gerber and Raymond [15] with MOF and ECore.

Since the goal of the transformations introduced in this paper is to translate between MLs defined with different MMLs, our technique usually needs a general-purpose transformation language. In this case, the recommended choice is XSLT given the extended availability of automated support, although the technique can be applied with other transformation languages.

3. Process for the Definition of a Metamodel Translation

The technique for defining metamodel translations contains the following steps:

Step 1: *Identification of correspondences between elements*. Find the element patterns that match with the source and target MMLs. Current MMLs are connection-based [22] MLs. Thus, all of these include concepts for nodes, edges that link nodes, and properties with primitive values.

Step 2: *Propagation of attributes and links*. Determine how to propagate the information and relations of nodes and edges from the source to the target metamodel. This information includes the properties of the elements in the metamodel.

Step 3: *Disambiguation of conflicts*. Look for elements in the target MML that are used to represent more than one element of the source MML. If any, the proposed technique places the results of these correspondences in different packages to clearly state the source of the translation. This organization reduces potential ambiguities in interpretation.

Step 4: *Implementation*. Define rules to transform the identified element patterns. Depending on the result obtained in the previous step, the target elements produced by the rules must be properly placed in the corresponding packages of the target.

The first step takes advantage of the common graph structure of current MMLs to establish correspondences between element patterns. It looks for modeling primitives that correspond to nodes, edges and properties. The concept of a node as an element that contains pieces of information usually offers clear correspondences amongst MMLs. A node is a *Class* in MOF, EMOF, and CMOF, an *EClass* in ECore, and an *Object* in GOPRR. The concept of edge presents more difficulties as the elements that it contains or the type of links are different among MMLs. For this reason, the number and type of elements required for its translation varies greatly amongst MMLs. Properties are usually described as attributes of some of the previous elements.

Depending on the features of MMLs, it is possible that the previous correspondences appear just between elements (i.e. correspondences 1-1) or between a group of elements (i.e. correspondences 1-n, n-1 or n-n). For very similar languages, the correspondences are 1-1. This is the case of the translation from CMOF to ECore that relates a CMOF *Class* with an ECore *EClass*, a CMOF *Association* with a pair of opposed ECore *EReference*, or a CMOF *Property* with an ECore *EAttribute* for the data of nodes. Note that these correspondences are so simple because ECore is almost aligned with EMOF. The only element from CMOF not included in EMOF in the previous translations is the *Association*, which also has a simple translation in ECore. Another possibility is that each source element can be translated to one element of the target, but not all the elements of the target MML are used. An example of this case occurs in the translation from ECore to MOF. These matchings are 1-1, so the translation is as simple as the aforementioned case. However, here it may be necessary to decide which elements of the target MML are used if several translations are possible. The translation of ECore to MOF also offers examples of translations n-1, where several ECore elements are translated to one MOF element. A third scenario is that in which the source elements contain more information than those of the target. In this case, each source element corresponds to a group of target elements that produce 1-n correspondences. This is the case for the GOPRR to MOF translation. A GOPRR relationship represents an edge, but several elements are required in MOF to represent that relationship. The GOPRR relationship has attributes and MOF needs at least one class to contain those attributes and several references to connect the linked nodes and the one that contains the information. The last case is n-n correspondences, where a group of elements of the target represents a group of elements from the source. An example of this situation is the translation from MOF to GOPRR. For instance, OMG's specifications contain recursive patterns for describing the edges: classes represent the body of the edge and their ends in the UML and SPEM metamodels. These OMG recursive patterns can be translated into a relationship and several roles in GOPRR.

The second step is intended to translate the information (represented by attributes and metattributes) and links from the source to the target metamodel. Attributes are information in the elements defined by the users. They are represented with *EAttributes* in ECore, *Properties* in MOF, or *Properties* in GOPRR. Metattributes are information in the elements defined as part of the MML. Examples of them in MOF are the multiplicity of the edge ends, which is represented with the *upperBound* and *lowerBound*, or indications of whether or not an attribute is required. The connections between the elements of the source are also translated into the connections of the target elements at this stage. Some examples of these connections are those between edges and nodes, or inheritance relations.

In Step 3, the designer of the translation must check if there is any target element used in more than one correspondence. This situation causes ambiguity in the translation, making the interpretation of the target model confusing and precluding traceability. When this occurs, this technique recommends placing the conflictive elements in different packages or another kind of container of the target metamodel to distinguish the source of their translations. Some examples of these conflictive translations appear from MOF to ECore and from MOF to EMOF. MOF have a greater expressive power than ECore or EMOF and thus some of their elements are translated into the same target elements. If there are conflicts, the designer establishes the structure of the target metamodels for the disambiguation. In general, structure-driven approaches implement this kind of conflictive translations with greater ease than a declarative one. Otherwise, a declarative approach is suitable for the translation.

Finally, in Step 4 the designer chooses a particular transformation language and creates the proper rules for the translation according to that language. The rules follow the matchings between source and target elements identified in Step 1 and propagate links and attributes according to Step 2. If Step 3 found conflictive situations, rules may need to place the target elements according to the structure established for the disambiguation of conflictive elements.

4. Case Studies

This section exemplifies the application of the technique described in the previous section with two case studies. They correspond to translations from GOPRR [10] to ECore [11] and from ECore to EMOF with the addition of *Associations* [6]. *Associations* are considered here, in spite of not belonging to EMOF but to CMOF, since there is no way to represent the concept of reference in EMOF. This section firstly makes a brief introduction to the concepts of these MMLs involved in the case studies. Then, it presents each case study with a motivation of the translation and the description of the application of the technique in that scenario.

4.1 Metamodeling Languages

4.1.1 GOPRR

GOPRR (Graph Object Property Relationship Role) [10] is the MML used in the MetaEdit+ [23] tool. It has also been used in frameworks for component reuse on metamodeling-based developments [24], to integrate concepts from different development stages (i.e. analysis, design and implementation) in object-oriented approaches [25], or in the adaptation of CASE tools to specific environments [26]. The main disadvantage of GOPRR is the lack of standardization in the instantiation of its metamodels, that makes the exchange of information difficult between tools using their proprietary formats. The main elements of GOPRR are:

- *Graph*. This is the root element which contains the other elements.
- *Object*. It represents an entity. The objects can extend other objects through inheritance hierarchies.
- *Relationship*. The relationships connect roles in any number. Therefore, the GOPRR relationships are n-ary.
- *Role*. It represents a relationship end. A role is connected to its *Relationship* and one *Object*.
- *Property*. It contains a value of a primitive type. Both objects, relationships and roles can have properties.

4.1.2 ECore

ECore is the MML of the *Eclipse Modeling Framework* (EMF) [11], [27], which is the metamodeling framework distributed as a plug-in of the Eclipse development environment. EMF provides support for the automated generation of textual editors from ECore metamodels and libraries for the serialization of both metamodels and models. With this support, the *Graphical Editing Framework* (GEF) [11] makes it possible to generate graphical editors for ECore metamodels. Also integrated in Eclipse as a plug-in, the ATL [16] defines transformations among ECore metamodels. ECore has a standard metamodel instantiation provided by EMF, in which, both metamodels and models are represented with XMI [28] documents. ECore includes as its key concepts:

- *EClass*. The single ECore element that contains EAttributes or *EReferences*. The instances of *EClass* can extend other instances of *EClass* by means of the *ESuperType* property.
- *EAttribute*. It contains values of primitive types, like integers, strings, and characters.
- *EReference*. It represents a binary relationship between two *EClasses*. The instance of the first *EClass* contains the instance of the *EReference*. The second *EClass* is referenced by the *EReference*. The *multiplicty* metatribute of *EReferences* is indicated by the *UpperBound* and *LowerBound* properties. There are two kinds of *EReferences*:
  - *containment*. The element representing the container includes an element that represents the referenced object. Each element can only be referenced by one containment reference.
  - *non-containment*. The container simply includes a path to the element representing the referenced element. An element can be referenced by several non-containment references.
- *EPackage, ESubPackage*. They are containers of *EClasses*. An *EPackage* can contain *ESubPackages*.

4.1.3 EMOF with *Associations*

EMOF is the core of the definition of the standard MOF [6].

GOPRR	ECore
Graph	EClass and containment EReferences
Object	EClass
Property	EAttribute
Relationship	EClass and containment EReferences
Role	EClass and non-containment EReferences

Fig. 1 Correspondences from GOPRR to ECore.

The key elements of EMOF are:

- *Class*. It is the element able to contain both Properties and Operations. It can extend other Classes, by means of the *Inheritance* concept.
- *Property*. It contains values of primitive types, such as integer, strings, and characters.
- *Package*. It represents a collection of related Classes and Associations.

For this case study we also consider the *Association* element from CMOF since EMOF does not contain an element to represent references. An *Association* describes a binary relation between classes where both association-ends are distinguished from each other by means of the *FirstElement* and *SecondElement* properties. The containment and non-containment Associations are distinguished from each other through the *IsComposite* property.

4.2 Translation from GOPRR to ECore

This section presents the translation from the GOPRR language to the ECore one. It offers examples of 1-n matchings and conflictive elements in the target metamodel. Thus, this translation requires a structure-driven approach according to the presented technique. The development of the translation follows the process proposed in Sect. 3, with the following subsections applying the different steps.

4.2.1 Correspondences between Elements

Step 1 requires determining the correspondences between GOPRR and ECore elements based on the identification of nodes, edges and their properties. For this purpose, the descriptions for GOPRR and ECore in Sect. 4.1.1 and Sect. 4.1.2 are considered. The correspondences detected in this step are introduced in Fig. 1 and described below.

Firstly, the *node* concept must be matched between GOPRR and ECore. A node in GOPRR is an *Object* that represents a piece of information and may contain *Properties*. The only element in ECore able to contain properties is an *EClass*. For this reason, the *EClass* element is selected as the translation for the *Object*.

In GOPRR and ECore, the properties of nodes are respectively represented with *Properties* and *EAttributes*. Both of them represent primitive values. This matching is another correspondence detected in this step.

A correspondence for the *edge* concept must also be found. The GOPRR *Relationships* and their *Roles* represent edges. As it happens with the GOPRR *Objects*, both *Relationships* and *Roles* can have properties. Since in ECore only the *EClass* element can contain properties, the translation of each *Relationship* and *Role* requires at least one *EClass*. Moreover, *Relationships* are connected to *Roles*. Thus, *Relationships* must also be represented with *EReferences* for their connections in ECore. Since each *Role* is only referenced by one *Relationship* in GOPRR, the corresponding *EReference* can be containment (see Sect. 4.1.2). In the same way, *Roles* are connected to *Objects*, so *Roles* must also be represented with *EReferences* besides *EClasses*. Each *Object* can be referenced by several *Roles* and for this reason, those *EReferences* related to *Roles* must be non-containment. Briefly, an *edge* is represented in GOPRR with a *Relationship* and several *Roles*. Its translation in ECore includes several *EClasses* (one for the *Relationship* and one for each *Role*) and several *EReferences* for the necessary connections (containment for the links between *Relationship* and *Roles* and non-containment for the links between *Roles* and *Objects*). Figure 2 presents an example of edge in ECore obtained as a result of the current translation from GOPRR. The example is taken from the INGENIAS [29] agent-oriented language.

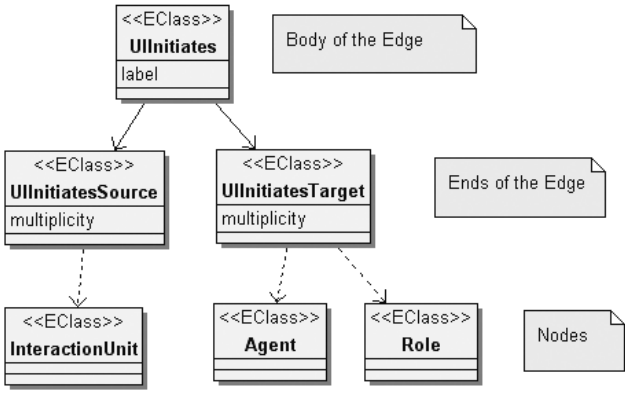

Fig. 2 An example of edge that is generated by the GOPRR-to-ECore taken from the INGENIAS ML. Boxes represent EClasses, continuous arrows containment EReferences, and discontinuous arrows non-containment EReferences.

Finally, in GOPRR the *Graph* is the container of the other elements. The correspondence of a *Graph* is established with an *EClass* that includes containment *EReferences* to link with the elements of that graph.

4.2.2 Propagation of Links and Attributes

Figure 3 introduces the correspondences for the propagation of links and attributes in the GOPRR-to-ECore translation according to Step 2. The links between GOPRR *Relationships* and *Roles* are preserved by means of containment *EReferences* between *EClasses*, as seen in the previous section. The containment *EReferences* are used to make sure that each *Role* is only referenced by one *Relationship*. In addition, the GOPRR *Roles* are connected to *Objects*, and these links are translated to non-containment *EReferences*. The reason is that the same *Object* can be referenced by sev-

GOPRR	ECore
Relationship - Roles	containment EReferences
Role - Objects	non-containment EReferences
Graph - Objects	containment EReferences
Graph - Relationships	containment EReferences
Graph - Roles	contained through the EClasses which are translated from GOPRR Relationships
link of GOPRR inheritance	ESuperTypes

Fig. 3 Propagation of links from GOPRR to EMF.

eral *Roles*. *Graphs* can contain *Objects*, *Relationships* and *Roles*. For *Objects* and *Relationships*, these links are preserved by means of containment *EReferences*, which allow an *EClass* to contain another. The links *Graph-Roles* cannot be translated to containment *EReferences*. ECore precludes that an *EClass* can be contained in more than one containment *EReferences*, and as seen before, the translations of *Roles* are already contained by the translations of *Relationships*. However, as the translation of *Graphs* actually contains the translation of *Relationships*, and these contain the translation of *Roles*, the translation of *Graphs* indirectly contains the translation of *Roles*.

Inheritance is a particular type of link. The GOPRR inheritance is translated into the inheritance of ECore, which is expressed with the metattribute called *ESuperTypes*. This translation preserves inheritance hierarchies.

As stated before, GOPRR properties are translated into ECore *EAttributes*. This step places these attributes in the elements that contain them. For example, an *Object* is transformed into an *EClass*; then the properties of the *Object* are transformed into *EAttributes* of the aforementioned *EClass*. In the same way, the properties of *Relationships* and *Roles* are also transformed and placed in their corresponding *EClasses*. According to the metattributes, the multiplicity of the connection between *Relationships* and *Roles* is translated in the ECore *LowerBound* and *UpperBound* properties of the corresponding containment *EReferences*.

4.2.3 Disambiguation of Conflicts

As indicated in the presentation of the technique (see Sect. 3), conflicts are detected by searching in the correspondences of the translation, which appear in Fig. 1. The *EClass* and *EReference* target elements are included in several correspondences, thus several conflicts arise. More specifically, *EClasses* and *EReferences* are included in the correspondences of the GOPRR *Graphs*, *Relationships* and *Roles*, and *EClasses* are also included in the correspondences of GOPRR *Objects*.

Therefore, according to the technique, conflicts must be disambiguated by a structure-driven approach. Firstly, the transformations must generate the target metamodel structure that permits the source of the elements to be distinguished. Considering the aforementioned conflicts, the structure of the target metamodel includes a package for each correspondence with conflict. Figure 4 shows this

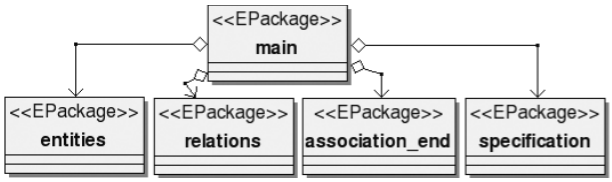

Fig. 4 The initial structure of the target metamodel of the GOPRR to EMF translation.

structure which contains the following packages:

- *entities*. This contains the *EClasses* that correspond to GOPRR *Objects*.
- *relations*. This contains the *EClasses* and *EReferences* that correspond to GOPRR *Relationships*.
- *association_ends*. This contains the *EClasses* and *EReferences* that correspond to GOPRR *Roles*.
- *specification*. This contains the *EClasses* and *EReferences* that correspond to GOPRR *Graphs*.

After generating the aforementioned packages, they are populated with the elements from the translations.

4.2.4 Implementation

As mentioned in Sect. 4.1.1, the serialization of metamodels described with GOPRR lacks a standard implementation. This work adopts the serialization used by the IDK [29] tool, which is introduced later. The IDK serializes the GOPRR metamodels as XMI [28] documents according to the following correspondences. Each *Object*, *Relationship* and *Role* is serialized as an XML element. These XML elements contain XML elements that represent the properties of their objects, relationships or roles. The XML element of an object also contains an *inherits* XML attribute to indicate the parent of the object. This attribute represents the inheritance among objects. The XML element of a relationship contains XML elements indicating the XML elements of its roles. Finally, the XML elements of the roles contain XML elements called *players* to indicate the valid objects that they connect.

The serialization of ECore has an accepted implementation provided by the EMF (see Sect. 4.1.2). This serialization represents ECore models as XMI documents.

Once the serialization formats of GOPRR and ECore have been chosen, designers implement the transformations. Section 2 already mentioned that this problem requires the use of a general-purpose transformation language since the transformations consider models defined with different MMLs. Among these transformation languages, the choice is XSLT as the metamodels are expressed in XML. XSLT has been widely used in model transformation literature [15], [30], [31]. The whole XSLT transformation for translating GOPPR metamodels into ECore metamodels is available on our web site [†]. An excerpt of this translation follows, where "..." omits parts of the transformation.

[†]http://grasia.fdi.ucm.es/ (in "Software"/"Additional Material for Papers" section)

```
<eSubpackages name="entities" ...>
  ...
  <xsl:for-each select=".../object">
  <xsl:element name="eClassifiers">
  <xsl:attribute ...> ecore:EClass </...>
  ...
<eSubpackages name="relations" ... >
  ...
  <xsl:for-each select=".../relationship">
  <xsl:element name="eClassifiers">
  <xsl:attribute ...> ecore:EClass</...>

<eSubpackages name="association_end" ...>
   ...
  <xsl:for-each select=".../role">
  <xsl:element name="eClassifiers">
  <xsl:attribute ...> ecore:EClass</...>
...
```

This excerpt of the translation shows the structure of the target metamodel. It includes the *ESubPackages entities, relations* and *association_end*. These subpackages contain the instances of the *EClass* element that are the conflictive element in this translation. The conflict arises from its use with nodes, edges and edge-ends that are respectively extracted from GOPRR *Objects, Relationships* and *Roles*. Each of these *EClass* instances are individually placed on the paths [†] `.../object`, `.../relationship` and `.../role` of the source metamodels.

The GOPRR to ECore translation presented has been successfully applied in the modernization of the model-driven development tool of the INGENIAS methodology [32]. The INGENIAS methodology is research work of our group aimed to develop multi-agent systems. Its main feature is the coverage of the whole development cycle, from analysis to implementation. Its tool support is known as the *INGENIAS Development Kit* (IDK) [29]. The specification of the INGENIAS ML has used metamodeling since its inception. One of the INGENIAS concerns is to provide a ML adaptable to the advances in agent research and for this reason the specification of its language and the development of its tools is based on metamodels. The INGENIAS metamodel currently has 88 objects, 89 relationships and 98 roles. The first versions of the metamodel [32] were expressed with GOPRR. In order to take advantage in the IDK of the automated support from the EMF, that metamodel was translated into ECore. This translation allows the IDK to use EMF [27] libraries for serialization or the future integration of ATL [33] in its model-driven development. Given the large number of elements in the INGENIAS metamodel, the automatic translation between MMLs saved an important amount of time and effort.

4.3 ECore to EMOF Translation

ECore and MOF are currently the most popular MMLs. The evolution pace of ECore is faster than that of MOF, as it is based on the Eclipse implementation of EMF while MOF is based on standardization. However, ECore is intended to be almost aligned (with minor differences) with EMOF. Therefore, translations between ECore and EMOF should help to bridge differences and at the same time make the differences explicit between both languages, while allowing the exchange of models between their users. As explained in the introduction to EMOF (see Sect. 4.1.3), it does not contain an element to represent references. The MOF element for references is the *Association* from CMOF, which is the extension of EMOF that defines the full MOF. To provide a complete translation of ECore, this case study also considers *Associations* in this section.

ECore	**EMOF with *Associations***
EClass	Class
EReference	Association
EAttribute	Property

Fig. 5 Correspondences between ECore and EMOF with *Associations*.

ECore	**EMOF with *Associations***
EClass - Ereference	firstElement of Association
EReference - EClass	secondElement of Association
ESuperTypes between EClasses	Inheritance of Classes

Fig. 6 Propagation of connections between ECore and EMOF with *Associations*.

Given the similarities, the translation between ECore and EMOF is simpler than the one presented in the previous section. It exemplifies an application of the technique with one-to-one correspondences and no conflictive target elements, since each target element is the result of only one matching. For the definition of the translation, this section considers the introductions to ECore in Sect. 4.1.2 and EMOF in Sect. 4.1.3.

Figure 5 introduces the correspondences between ECore and EMOF languages according to Step 1 of the technique. Nodes correspond to the ECore *EClass* and EMOF *Class* elements and the edges to the ECore *EReference* and to CMOF *Association*. Notice that in this translation, the references represent the edges because in the source and in the target MMLs there is the same information. Neither ECore *EReferences* nor CMOF *Associations* contain attributes, and their metattributes are similar to each other.

Step 2 focuses on the propagation of information and links. The metattributes in EMOF and ECore are very similar and thus the *copy* actions are simple. The connections between elements (see Fig. 6) are also copied from the source metamodel to the target metamodel. In this case, the *ESupertypes* of *EClasses* and the connections of *EReferences* are respectively copied to the inheritance of *Classes* and connections of *Associations* in CMOF. In ECore, the connection between *EClasses* and their *EReferences* is different from the connection between *EReferences* and the refer-

[†] "..." omits parts of the paths.

enced *EClasses*. This difference also exists in CMOF, and it is indicated by the *firstElement* and *secondElement* metattributes of *Associations*.

In this translation, none of the target elements is produced by more than one correspondence. Thus, a declarative approach is enough to define the transformations according to Step 3.

Finally, in Step 4, designers define declarative rules so each source element is translated into its corresponding target element following the one-to-one correspondence. This translation is defined with XSLT [21] for experimentation, but any other transformation language that support the declarative approach could be used. This translation can be symmetrically defined in the two directions, from ECore to EMOF, and from EMOF to ECore.

5. Related Work

Literature offers examples on translations between different MMLs that can be classified into three main groups: the use of general-purpose programming languages for translations, general-purpose transformation languages, or transformation languages for specific metamodels.

The use of general-purpose programming languages [17] relies on the availability of APIs to internal representations of the models of the considered MMLs. The translation itself is programmed in these languages as a manipulation of these internal models. Examples of this approach are several MDE tools that include code generation, as this approach is well-suited for this kind of instantiation. Among these tools appear OptimalJ [19], the IDK [29] support to build modules, AndroMDA [34], or MS/DSL [13]. This approach has the advantage of efficiency and a high flexibility as designers are only tied by the limitations of the programming language. In terms of drawbacks, it lacks scalability given that the support for MMLs, MLs, and transformations is always built ad-hoc. Besides, it hides and intermingles most of this information in the code, making it difficult to understand.

The second group of approaches relies on general-purpose transformation languages for the translation. One work to consider here is the aforementioned one by Gerber and Raymond [15] that defines a translation between MOF and EMF (i.e. ECore in this work) using XSLT. The resulting XSLT transformations are difficult to maintain and understand given their verbosity and complexity. The MTRANS framework [30] tries to overcome these limitations by adding a layer of declarative rules that are later translated to XSLT. This line of research is closer to ours but it does not describe a detailed technique for defining metamodel translations. Besides, the work in [15] only focuses on a case where the MMLs are almost aligned. As seen in the case study with GOPRR, more complex techniques are required when the MMLs are different. Nevertheless, our work could use a framework like MTRANS to ease the implementation of the transformations in Step 4.

The third group of approaches uses transformation languages for specific metamodels. The ATLAS Group has carried out several projects to translate between ECore and other MMLs like GME [35] or MS/DSL [36] using its ATL [16]. These works take advantage of the possibility of generating XMI files from non-ECore models through XML injectors. They then define an ECore metamodel for the non-ECore MMLs and use ATL transformations to generate models compliant with the new ECore metamodels from the XML files. Additional modifications can be made with standard ATL mechanisms. These works offer some hints on how to build the new ECore metamodels but they are not as detailed as the technique guidelines in this paper. Besides, these translations split the interpretation of the original language between the development of the XML injectors, the creation of the new metamodels, and the ATL transformations. Although one could argue that this process offers a certain degree of modularity, it makes it difficult to follow the design decisions or to find mistakes. Our approach, on the other hand, prefers to work with one translation resource that centralizes the interpretation and the transformation.

From a broader perspective, all these efforts focused on MMLs translations can be seen as part of the effort to enable the exchange of information between different modeling and technical spaces. According to [37], a *modeling space* is a modeling architecture defined by a particular meta-metamodel, while a *technical space* is a working context with a set of associated concepts, body of knowledge, and tools. Examples of modeling spaces are those of MOF [6] or EBNF [38], and of technical spaces are those related with XML [39], MDA [6], or EMF [12]. Research in this field [37], [40] proposes the use of general projectors (either injectors or extractors) to propagate information to those spaces where it has to be processed. Nevertheless, this is still a recent theoretical work without implementations.

6. Conclusions and Future Work

This paper presents a guideline to define translations between metamodels defined with different MMLs. This technique is based on the common basis of these MMLs in graph representations. Thus it proposes the definition of a catalogue of mappings between MMLs that relates the primitives used to represent the different types of nodes, edges, and their properties in these languages. The technique then identifies new correspondences that enable the propagation of links between the previous elements and of their attributes. Finally, in the case of conflicts in correspondences, the technique proposes the addition of additional packages for disambiguation. The conflicting elements in the target metamodel are rearranged in the packages in such a way that no pair of conflicting elements are in the same package. The definition of rules for the kind of rearrangement is easier with a structure-driven approach. If there are no conflicts, the technique adopts a declarative approach.

In order to illustrate the technique and its complexity,

[†]http://grasia.fdi.ucm.es (in “Software” section)

the paper includes two case studies. They clarify some important semantic and syntactic differences between available MMLs outside the MOF space (in the GOPRR case) and the case of issues that can arise even with similar languages (like ECore and EMOF).

The work presented is an initial step in addressing the development of translations between different modeling spaces. Until now, our research has focused on MMLs, where we plan to extend the development of translations to other MMLs such as KM3, Coral or the full MOF. This work will allow the exchange of information between different platforms, but also a detailed analysis and clear comparison regarding the features of these MMLs. Another important issue is the need to adopt a model-driven approach for the development of transformations. As seen in the experimentation and related works, developing these general-purpose transformations is error prone, generates cumbersome transformations, and makes maintenance difficult. A model-driven approach should provide an abstraction layer over general-purpose transformation languages, in the line of work with MTRANS. More specifically, our research group has begun the development of a tool that generates transformations from models. A prototype of this tool is now available for practitioners in GRASIA web [†].

Acknowledgements

This work has been supported by the project *Methods and tools for agent-based modeling* supported by the Spanish Council for Science and Technology with grant TIN2005-08501-C03-01, and by the grant for Research Group 910494 by the Region of Madrid (Comunidad de Madrid) and the Universidad Complutense Madrid.

References

[1] A. Rugnone, C. Nugent, M. Donnelly, D. Craig, C. Paggetti, E. Tamburini, and E. Vicario, "HomeTL: A visual formalism, based on temporal logic, for the design of home based care," IEEE Int. Conf. Automation Science and Engineering, CASE 2007, pp.747–752, 2007.

[2] J. Tolvanen and M. Rossi, "MetaEdit+: Defining and using domain-specific modeling languages and code generators," Conference on Object Oriented Programming Systems Languages and Applications, pp.92–93, 2003.

[3] D. Amyot, H. Farah, and J. Roy, "Evaluation of development tools for domain-specific modeling languages," Proc. 5th Workshop on System Analysis and Modelling, vol.4320/2006, pp.183–197, 2006.

[4] K. Balasubramanian, J. Balasubramanian, J. Parsons, A. Gokhale, and D. Schmidt, "A platform-independent component modeling language for distributed real-time and embedded systems," J. Comput. Syst. Sci., vol.73, no.2, pp.171–185, 2007.

[5] D. Schmidt, "Guest Editor's introduction: Model-driven engineering," Computer, vol.39, no.2, pp.25–31, 2006.

[6] OMG, "Meta Object Facility (MOF) core specification. Version 2.0." http://www.omg.org/ (available on Aug. 15, 2008), Jan. 2006.

[7] OMG, "OMG Unified Modeling Language (OMG UML), Infrastructure, V2.1.2." http://www.omg.org/ (available on Aug. 15, 2008), Nov. 2007.

[8] OMG, "OMG Unified Modeling Language (OMG UML), supestructure, V2.1.2." http://www.omg.org/ (available on Aug. 15, 2008), Nov. 2007.

[9] F. Jouault and J. Bezivin, "KM3: A DSL for metamodel specification," Formal Methods for Open Object-Based Distributed Systems, Lecture Notes in Computer Science, vol.4037/2006, pp.171–185, 2006.

[10] S. Kelly, GOPRR Description, PhD Dissertation, vol.Appendix 1, Jyväskylä University, Finland, 1997.

[11] B. Moore, D. Dean, A. Gerber, G. Wagenknecht and P. Vanderheyden, Eclipse Development using Graphical Editing Framework and the Eclipse Modelling Framework, IBM Redbooks, 2004.

[12] Eclipse, "Eclipse Modeling Framework (EMF)." http://www.eclipse.org/modeling/emf/ (available on Aug. 15, 2008).

[13] J. Greenfield and K. Short, Software factories: Assembling applications with patterns, models, frameworks and tools, ACM Press New York, NY, USA, 2003.

[14] M. Alanen and I. Porres, "Coral: A metamodel kernel for transformation engines," Proc. 2nd European Workshop on Model Driven Architecture (MDA 2004) with an emphasis on Methodologies and Transformations, University of Kent, United Kingdom, Technical Report.

[15] A. Gerber and K. Raymond, "MOF to EMF: There and back again," Proc. 2003 OOPSLA Workshop on Eclipse Technology eXchange, pp.60–64, 2003.

[16] F. Jouault and I. Kurtev, "Transforming models with ATL," Proc. Model Transformations in Practice Workshop at MoDELS, vol.3844, pp.128–138, 2005.

[17] K. Czarnecki and S. Helsen, "Classification of model transformation approaches," Proc. 2nd OOPSLA Workshop on Generative Techniques in the Context of the Model Driven Architecture, 2003.

[18] A. Gerber, M. Lawley, K. Raymond, J. Steel, and A. Wood, "Transformation: The missing link of MDA," Graph Transformation: First International Conference, Icgt 2002, Barcelona, Spain, Oct. 7-12, 2002: Proceedings, vol.2505/2002, pp.90–105, 2002.

[19] Compuware, "OptimalJ 3.0 user's guide." http://www.compuware.com/products/optimalj (available on Aug. 15, 2008).

[20] OMG, "Meta Object Facility (MOF) 2.0 query/ view/ transformation specification. Version 1.0." http://www.omg.org/ (available on Aug. 15, 2008), April 2008.

[21] W3C, "XSL Transformations (XSLT) Version 1.0." http://www.w3.org/TR/xslt (available on Aug. 15, 2008), Nov. 1999.

[22] G. Costagliola, A. Delucia, S. Orefice, and G. Polese, "A classification framework to support the design of visual languages," Journal of Visual Languages and Computing, vol.13, no.6, pp.573–600, 2002.

[23] S. Kelly, K. Lyytinen, and M. Rossi, "MetaEdit+ a fully configurable multi-user and multi-tool CASE and CAME environment," Advanced Information Systems Engineering: 8th International Conference, CAiSE'96, Heraklion, Crete, Greece, May 20-24, 1996: Proceedings, vol.1080/1996, pp.1–21, 1996.

[24] Z. Zhang and K. Lyytinen, "A framework for component reuse in a metamodelling-based software development," Requirements Engineering, vol.6, no.2, pp.116–131, 2001.

[25] J. Van Hillegersberg and K. Kumar, "Using metamodeling to integrate object-oriented analysis, design and programming concepts," Information Systems, vol.24, no.2, pp.113–129, 1999.

[26] K. Lyytinen, P. Marttiin, J. Tolvanen, M. Jarke, K. Pohl, and K. Weidenhaupt, "CASE environment adaptability: Bridging the islands of automation," Proc. 8th Annual Workshop on Information Technologies and Systems (WITS'98), pp.1–13, 1998.

[27] F. Budinsky, Eclipse Modelling Framework: Developer's Guide, Addison Wesley, 2003.

[28] OMG, "MOF 2.0/XMI mapping, Version 2.1.1." http://www.omg.org/ (available on Aug. 15, 2008), Dec. 2007.

[29] G. of Research on Agents: Software Engineering and A. (GRASIA), "INGENIAS development kit." http://ingenias.sourceforge.net/ (available on Sept. 2, 2008).

[30] M. Peltier, J. Bézivin, and G. Guillaume, "MTRANS: A general framework, based on XSLT, for model transformations," Workshop on Transformations in UML (WTUML), Genova, Italy, April 2001.
[31] R. Grønmo and J. Oldevik, "An empirical study of the UML model transformation tool (UMT)," Proc. First Interoperability of Enterprise Software and Applications, Geneva, Switzerland, Feb. 2005.
[32] J. Gómez-Sanz and J. Pavón, "Agent oriented software engineering with INGENIAS," Multi-Agent Systems and Applications III, Lecture Notes in Artificial Intelligence, vol.2691, pp.394–403, 2003.
[33] F. INRIA (Institut National de Recherche en Informatique et en Automatique), "ATL home page." http://www.eclipse.org/m2m/atl/ (available on Aug. 15, 2008).
[34] A. Team, "AndroMDA-4.0." http://www.andromda.org/ (available on Aug. 15, 2008), May 2007.
[35] J. Bézivin, C. Brunette, R. Chevrel, F. Jouault, and I. Kurtev, "Bridging the Generic Modeling Environment (GME) and the Eclipse Modeling Framework (EMF)," Proc. Best Practices for Model Driven Software Development at OOPSLA, vol.5, 2005.
[36] J. Bézivin, G. Hillairet, F. Jouault, I. Kurtev, and W. Piers, "Bridging the MS/DSL tools and the eclipse modeling framework," Proc. International Workshop on Software Factories at OOPSLA, 2005.
[37] D. Djurić, D. Gašević, and V. Devedžić, "The Tao of modeling spaces," J. Object Technology, vol.5, no.8, pp.125–147, 2006.
[38] I. JTC1, Information technology - Syntactic metalanguage - Extended BNF, ISO/IEC Internationl Standard, vol.14977:1996 (E), ISO/IEC, 1996.
[39] W3C, Extensible Markup Language (XML) 1.0, Fourth ed., 2006.
[40] J. Bézivin, V. Devedžić, D. Djurić, J. Favreau, D. Gašević, and F. Jouault, "An M3-Neutral infrastructure for bridging model engineering and ontology engineering," Proc. 1st International Conference on Interoperability of Enterprise Software and Applications, pp.159–172, 2005.

Iván García-Magariño was born in 1982, in Madrid. He is a Ph.D. student of Computer Science at the Complutense University in Madrid, Spain. He is currently working with the GRASIA research group (http://grasia.fdi.ucm.es/). He has also published articles in the "International Transactions on Systems Science and Applications" journal and conferences such as Autonomous Agents and Multiagent Systems (AAMAS'08), Agent-Oriented Software Engineering (AOSE'08), Practical Applications of Agents and Multi-Agent Systems (IWPAAMS'07). He and other members of his research group won the prize of "Best Academic Demo" at the AAMAS'08. His 14 publications over the last year relate to model-driven engineering, metrics, and multi-agent systems. He worked as a software consultant before joining the GRASIA group. His work is now supported by a research grant at the Complutense University in Madrid.

Rubén Fuentes-Fernández holds a Ph.D. in Computer Science from the Complutense University in Madrid, Spain. He is Associate Professor at this university and member of the GRASIA research group. Amongst other journals, he has had research papers published in "IEICE Transactions on Information and Systems", "IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans", "International Journal of Web Engineering and Technology", and "International Journal of Agent-Oriented Software Engineering". Previously, he worked as a consultant in database systems for four years. His main interests are concerned with the application of Social Sciences to software development, model-driven engineering, and agent-oriented methodologies.