GraCoT, a tool for co-creation of models and metamodels in specific domains

Paola Gómez
Universidad de los Andes
Department of Systems and Computing Engineering
Bogotá, Colombia
pa.gomez398@uniandes.edu.co

Mario Sánchez
Universidad de los Andes
Department of Systems and Computing Engineering
Bogotá, Colombia
mar-san1@uniandes.edu.co

Jorge Villalobos
Universidad de los Andes
Department of Systems and Computing Engineering
Bogotá, Colombia
jvillalo@uniandes.edu.co

ABSTRACT
In many domains, models are created based on predefined metamodels which abstract the structure of the domain in question. However, there are specific domains, like Enterprise Architecture (EA) projects, where a metamodel cannot be defined in advance to the creation of the model. Unfortunately, in this situation using standard frameworks, like EMF, generates some inconveniences in the construction of the model and the metamodel because these frameworks do not support the manipulation of metamodels at runtime. In this paper, we propose a strategy to co-create metamodels and models in an incremental and simultaneous way. This proposal is supported by a dynamic approach that separates the linguistic and the ontological conformity concerns of metamodeling. This strategy has been implemented in a graphical editor called GraCoT, which also provides interactive assistance to guide the users during the co-creation process.

Categories and Subject Descriptors
D.2.2 [Software Engineering]: Design Tools and Techniques; D.2.4 [Software Engineering]: Software/Program Verification—Model checking; D.2.6 [Software Engineering]: Programming Environments—Graphical environments; D.2.11 [Software Engineering]: Software Architectures—Domain-specific architectures

General Terms
Design, verification

Keywords
Co-creation, metamodel, model, dynamic validation of conformity, conformity problems, interactive user assistance, specific domains

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ACME ’13, Montpellier, France
Copyright 2013 ACM 978-1-4503-2036-8 ...$15.00.

1. INTRODUCTION
Typically, the process of models and metamodels construction follows a chronological order where the metamodel must be created before the model. A metamodel abstracts the concepts and relations of a specific domain and represents the structure of this specific domain defining what can and cannot be expressed in the models. For instance, Figure 1 shows a metamodel where the concepts Enterprise, Employee and Client are defined. This metamodel establishes that an Enterprise can have several employees and clients, and that each Employee is related with at least one Client. The model shown in the lower part of Figure 1 is said to conform to the metamodel because its structure respects the established rules: an enterprise has two employees and three clients, and each employee is related to at least one client.

However, in some domains the usual sequence of construction (first metamodel, then model) cannot be applied because the metamodel can change after a model is created. Particularly, in the Enterprise Architecture (EA) context, when a project starts, only a partial metamodel exists, and it is progressively completed, refined, and adapted as the model is constructed.

In EA projects, the main goal is to build a model of the enterprise that relates business elements to the informational...
and technological aspects [14, 16]. The model is then analyzed and used to support decisions regarding problems of the enterprise. In this context, the model is used to analyze the structure of the enterprise, and especially the relations between business, information, and technology elements. Because of the plurality of concerns, these models tend to be large and complex. So does the metamodel.

During the run of an EA project, when new valuable information is found, or when the focus of the project changes, both the metamodel and the model must change in an incremental and simultaneous way [12]: changes to the metamodel usually require changes to the model; similarly, some changes to the model can only be supported if the metamodel changes as well. This situation has been called co-creation of models and metamodels.

When standard modeling frameworks are used, such as EMF [1], the usual first metamodel then model strategy is the only one allowed. This happens because these frameworks are based on a strong conformance relation where the model must conform to the metamodel in every moment. Any change in the metamodel or the model can break the conformity relation unless the other artifact is also changed accordingly, which is very problematic when metamodels cannot be changed at runtime, such as in the case of EMF. As a result, attempting to use these frameworks for co-creating models and metamodels is difficult and involves a large number of technical problems to solve. In order to avoid this problem, a frequent strategy is to use tools such as word processors or spreadsheets to handle the model information in a non-structured way. This leads to many other problems such as inconsistencies and expensive maintenance.

The aforementioned problems are illustrated in Figure 2. In case A, the model of the enterprise initially conforms to a metamodel that defines the concepts Enterprise and Employee and establishes a relation between them; the model has initially two employees related to the enterprise. Later on, some changes are introduced in the model when two new clients are associated to the enterprise. In this moment the conformance is broken because the metamodel did not define the concept Client or its relation with the enterprise.

In case B, the initial model and metamodel of case A are used. Now however, changes are introduced in the metamodel: the concept Client is introduced, as well as one mandatory relation establishing that each employee must have at least one client. This last change breaks the conformance of the model as in case A.

This problematic situation could be solved by tools offering the following capabilities. Firstly, it should be possible to introduce changes in the metamodel as a response to changes in the model that break the conformance. Secondly, it should be possible to trace the effect on the models of changes to the metamodel.

In this paper we present GraCoT (Graphical Co-creation Tool), a graphical editor for models and metamodels which is based on GMF [6] and other Eclipse technologies. GraCoT also provides interactive assistance to guide users during the co-creation process.

Moreover, GraCoT handles the metamodels in a dynamic way, avowing the limitations of EMF as a result. Thus, GraCoT separates the linguistic and ontological conformance concerns, following Kühne’s approach [13]. Furthermore, it uses a generic intermediate metamodel called GIMM (Generic Intermediate Metamodel) to support the linguistic conformance relation.

The rest of the paper is structured as follows. Section 2 presents in detail the solution strategy. Section 3 presents GraCoT in detail (Architecture, GUI, the validation engine, and the interactive user assistance mechanism). Section 4 illustrates with an example the use of GraCoT in a co-creation process. Section 5 briefly presents some related work. Finally, Section 6 presents the conclusions.

2. A STRATEGY FOR CO-CREATION

The main goal behind the design of GraCoT as a tool to support co-creation processes was to enable the modification of metamodels at runtime. This was achieved by distinguishing between linguistic and ontological conformance. From this original goal, the following secondary goals were derived: the need to evolve validation rules as metamodels are modified; the need to reduce the complexity of the co-creation process for users by providing interactive assistance; and the need for compatibility with other tools by means of standard formats. This section presents a summary of the strategy used in GraCoT to support the aforementioned goals. The next section presents how the strategy was implemented in GraCoT.
2.1 Linguistic and ontological conformance

According to Kühne’s approach [13], metamodeling can be separated in an ontological and a linguistic dimension, which must be considered simultaneously. The ontological dimension considers metamodels to describe the elements of the reality and the valid ways to relate them. We call these domain metamodels, and their nature is dynamic because they can change at any time during the modeling process. On the other hand, the linguistic dimension considers metamodels to define just the structure of the models (elements, attributes, and relations), independently from the domain. Considering both dimensions simultaneously, each model instance is at the same time an ontological instance and a linguistic instance. This approach is used by [10] in order to allow ontological modeling with an arbitrary number of meta-levels.

In frameworks such as EMF, this separation is not supported. Therefore, the model conformity is validated against the same metamodel combining the two dimensions. Although this is not necessarily inconvenient, it results in a very static metamodeling process, and it does not allow changes to the metamodels at runtime.

In order to support the linguistic dimension of conformance, we propose an intermediate generic metamodel called GIMM, which provides the necessary meta-types to define any model. We define GIMM’s nature as static because it never changes.

In order to separate the two conformance dimensions, we propose to consider, for each model, two different metamodels during the whole modeling process. On the one hand, there is the domain metamodel which is likely to evolve during the co-creation process. The conformance between the model and this domain metamodel has to be handled in a very dynamic way that can adjust immediately to the changes introduced. On the other hand, there is a static linguistic metamodel that only defines the primitives and basic rules to define generic models. The means to validate the conformance between the model and the linguistic metamodel are only syntactical in nature and not expected to ever change.

To illustrate this point, we present GIMM, the linguistic metamodel used in GraCoT. It will be later shown that every model constructed using GraCoT is linguistically an instance of GIMM, and is ontologically an instance of the corresponding domain metamodel.

Figure 3 shows GIMM, which was inspired by the subset of the UML metamodel that serves to describe object diagrams. We decided not to use the Ecore Metamodel as intermediate metamodel for simplicity. GIMM is implemented by means of the traditional EMF mechanisms. Thus, a framework of classes (EClass) based on this metamodel was generated and is used for the construction and validation of the model.

The root of GIMM is called Model and serves as a container for all other elements. The meta-types Element and Relation serve to represent, respectively, the element instances that appear in a model, and the relationships between them. The meta-type Relation has been classified in ContainmentRelation and CrossRelation to distinguish conceptually the containment property of one EReference in the domain metamodel. The type Attribute serves to represent the information about the attributes of the elements contained in the model. Each Attribute has a typeName, a datatype, and a value. The datatypes provided by GIMM support integers, doubles, strings, booleans, and dates.

The attribute typeName of the meta-types Element, Relation serves to relate each instance of the model with the corresponding meta-type in the domain metamodel. This will be further explained later on the paper.

2.2 Dynamic validation of conformity

To identify conformity problems against the domain metamodel, it was necessary to have an engine capable of verifying the ontological conformance. This engine has to match the typeName of each instance in the model with the corresponding meta-type in the domain metamodel in order to identify valid instances and verify additional ontological aspects. In addition, the engine is capable of verifying the linguistic conformance respects to GIMM to guarantee several structural aspects needed to verify the ontological conformance.

The key factor about the validation engine is that it has to be dynamic. This means that it should be capable of supporting changes to the domain metamodel and adapting the validation rules to the present structure of the metamodel.

2.3 User guidance

To solve the problems that motivate this work, it is also necessary to guide users while solving conformity problems. For each problem appearing during a co-creation process, there may be several solutions, with many potential consequences. It is desirable to guide the user by offering the alternative solutions and giving adequate explanations. These alternative solutions may require modifications to the model or the domain metamodel, and sometimes these may require additional information to apply the changes.

2.4 Standard formats

The strategy proposes to guarantee compatibility with standard tools in order to manipulate different standard formats and provide versatility to the solution. This decision was given because any model of the solution is an instance
of GIMM; therefore, none of the current tools could manipulate this model.

3. GRACOT

The strategy described in the previous section has been implemented in a graphical editor based on EMF and GMF called GraCoT. According to the strategy, GraCoT separates the validation of the linguistic conformance (against GIMM) and the ontological conformance (against a domain metamodel). As a result, GraCoT offers a suitable tool to support co-creation processes, where metamodels can be modified even after the modeling phase has started. Furthermore, during these processes the tool provides assistance to the users. This section presents GraCoT from a number of perspectives: architecture of the tool, elements of its GUI, validation mechanisms, user assistance methods, and capabilities to handle standard exchange formats.

3.1 Architecture

The architecture of GraCoT is presented in Figure 4. The tool is based on the Eclipse platform, and its architecture includes a number of plug-ins. Some of these form the core component of the tool, while others just provide extension points to allow their interaction. The seven components of GraCoT will now be described.

![Figure 4: GraCoT architecture](image)

EuGENIa Editor – EE: this component provides the graphical interface of GraCoT that serves to create model diagrams (see section 3.2). This component is an immutable EuGENIa [5] editor that was created around the GIMM metamodel. As a result, the models created with this editor always conform to GIMM, and the standard verification mechanisms can be used. In order to support all this, the EE component includes the GIMM metamodel as well.

The EE is related to the following components: Rules Validation – RV, and Wizards – W. RV provides EE with information about problems in the model which have to be marked as warnings or errors. W provides user assistance tools, which are associated to the different kinds of problems, and options to import and export the models from and to xmi files.

Rules Validation – RV: this component verifies the ontological conformance of any model with respect to the domain metamodel. RV operates in a dynamic way, which means that the validation rules change every time the domain metamodel changes. It is implemented on top of EVL (Epsilon Validation Language) technology [4] and provides extension points to support the interaction with EE. Section 3.3 provides more details about this component.

 Updates Generation – UG: this component regenerates the validation rules that RV uses. This regeneration process is activated when the user wants. As a result, the validation of ontological conformance is consistent at all times with respect to last regeneration done by the user. By the interaction between UG and RV, it can be said that GraCoT has a dynamic validation engine.

Wizards – W: this component provides a number of wizards that support the requirement for user assistance. This assistance is interactive and targets the resolution of conformance problems in the following way: when a change is made to the model in the EE, the RV checks for problems in the model; if one is found, the RV reports the kind of the problem and the EE requests the W for alternative wizards that can be used to solve the problem; after the user selects one of the offered wizards, he has to follow the proposed steps and provide the information required in order to regain conformance. When the changes are applied, the model is checked again to review new possible problems. In addition, whether the changes are applied over the domain metamodel, W informs the UG component in order to regenerate the validation rules.

Additionally, W provides the extension points used by EE to offer general GraCoT capabilities such as import and export to xmi, or to replace the domain metamodel.

Transformations – TR: this component provides the mechanisms to change a standard xmi model to a GIMM model and vice versa. This component provides compatibility with EMF-based tools, and makes it possible to import existing models and to export models that are the result of co-creation processes.

### Table 1: Architecture components and technologies

<table>
<thead>
<tr>
<th>Component</th>
<th>Technology</th>
</tr>
</thead>
<tbody>
<tr>
<td>EuGENIa Editor</td>
<td>Eugenia, GMF</td>
</tr>
<tr>
<td>Rules Validation</td>
<td>EVL, EOL [3]</td>
</tr>
<tr>
<td>Updates generation</td>
<td>EMF, Xpand [8]</td>
</tr>
<tr>
<td>Transformations</td>
<td>EMF</td>
</tr>
<tr>
<td>Fusion</td>
<td>EMF</td>
</tr>
<tr>
<td>Wizards</td>
<td>SWT, JFace[7]</td>
</tr>
</tbody>
</table>

Instructions for installing GraCoT are available at: http://gracot.virtual.uniandes.edu.co/index.php/download

GraCoT web site: http://gracot.virtual.uniandes.edu.co/
Fusion Utilities – FU: is a component that is part of the Fusion Project [2]. Using this component, it is possible to manipulate and to query the domain metamodel.

Utilities – U: this component offers support for generic, simple operations, such as handling property files and apply format validation to data values.

The architecture described was constructed using several technologies and frameworks. The technologies that support each component are summarized in table 1.

3.2 GUI

This section presents GraCoT’s GUI and the general features it provides. Figure 5 presents a screenshot of the graphical interface, with its main elements framed. The upper-left area shows the canvas to create GIMM models. The appearance of this graphical editor was tweaked in order to make the diagrams use the syntax of UML’s object diagrams. Note that the attributes appearing inside objects are those specified for the corresponding meta-type in the domain metamodel. Nevertheless, this model editor does not really know about the domain metamodel, and the consistency of the diagram is guaranteed by the \( RV \) component that was previously described.

The canvas has a palette that offers the tools to create instances inside the models. Moreover, these tools do not depend on the domain metamodel, which in some cases may include hundreds of meta-types and thus render the palette impossible to use. Instead, the tools in the palette correspond to the members of GIMM and make it possible to create entities (EClass), attributes (EAttribute), and containment and non-containment relations (EReference).

The upper-right area of Figure 5 shows the domain metamodel. This is an unmodified GMF graphical editor, and it makes it possible to manipulate the metamodel directly. When changes to the metamodel are saved and the updating rules are activated by the user, GraCoT is modified accordingly (validation rules, wizards, errors and warnings depicted in the editor).

The bottom-left area of the Figure presents a regular Eclipse properties view which presents information about the instance currently selected on the model editor. Since this editor does not know about the domain metamodel, the properties that appear are those of the corresponding GIMM types. The domain properties have to be modified directly on the diagram.

The problems view, shown on the bottom right side of the Figure, presents the details about the problems identified in the GIMM model. The problems detected are classified as errors or warnings, and they are also marked on the canvas. When one problem is marked as a warning, it means that the problem has at least one suggested solution associated. These solutions can be accessed through the quick fix option.

Finally, GraCoT’s GUI provides other features that can be found in the File menu (e.g., create diagram models or export GIMM model).
3.3 Validation engine

The validation engine designed for GraCoT is responsible for verifying the ontological and linguistic conformity of the GIMM model. This verification is based on rules where each one checks some possible inconsistency over a particular instance meta-type of the model.

3.3.1 Validation algorithm

EVL, the technology used to develop the validation of models, defines contexts related to specific metamodel elements. Each context can have several invariants that check conditions over the corresponding instances in the model. In GraCoT, each invariant is called a validation rule.

The definition of contexts and invariants in the right order is essential for the successful execution of the validation. As a result, it is possible that some invariants require other invariants in the same or different context to be evaluated before. When the execution of a rule is successful, the result is associated with the corresponding context, and inconsistencies are marked in the corresponding meta-type instance. The priority order for the evaluation of contexts in GraCoT is the following: 1. Model, 2. Element (EC), 3. Attribute (AC), 4. ContainmentRelation (CRC) and 5. CrossRelation context (CrRC). Additionally, there are dependencies between certain rules that make it necessary to evaluate some before the others. These dependencies will be discussed and shown in detail in the next section.

Depending on the kind of context and the specific rule, validation rules can apply identically to all of the elements in the context, or they can apply with slight variations to each element. As an example of the former case, consider the validation rule that verifies the attribute typeName of each Element. In each case, the value has to match the name of some meta-type in the domain metamodel.

In the latter case, rules have to be replicated with some variation. For example, there is a rule that verifies that the attributes within an Element correspond to those defined in the domain metamodel for the corresponding meta-type. The capabilities of EVL make it necessary to generate alternative versions of this rule (one per meta-type), and evaluate each alternative only in the context of certain elements (those that are instances of the corresponding meta-type).

3.3.2 Validation Rules

The EVL invariants or validation rules used by the validation engine are dynamically generated by the UG component based on Xpand templates. When the domain metamodel changes, or when a new domain metamodel is loaded, these templates are used to re-generate the validation rules. Immediately afterwards, the new validation rules are used and all the warnings and problems displayed in the canvas and the problems view are updated.

Currently, GraCoT supports 31 kinds of validation rules that can be classified into two groups: linguistic rules and ontological rules. The group of linguistic rules is composed by 15 rules checking that the model is properly constructed with respect to the GIMM and to some additional restrictions. For example, one of the rules checks that the value of the attribute typeName is properly formed (e.g., no blanks, no symbols) for each entity. The problems that can be detected by applying these linguistic rules are presented in table 2.

The second group of rules includes 16 ontological rules to evaluate the ontological conformance of the model to the domain metamodel. For instance, one of the rules verifies that the attribute typeName of each Element matches some EClass name in the domain metamodel. The problems that can be detected by applying these ontological rules are presented in table 3.

For most of the rules discussed it is possible to offer at least one alternative solution for the problems detected. In those cases, the problems are marked as warnings both in the canvas and the problems view. For example, when the attribute typeName of one Element instance is invalid, there are two possible solutions. The first one is to modify the domain metamodel and create a new meta-type with the proposed name; the second one is to remove the infringing Element instance from the model. The alternative solutions are indicated in each rule, and some of them are directly specified in the EVL. In other cases, other components are responsible for providing the necessary solutions (e.g., wizards).

Some of the validation rules evaluate whether mandatory information exists in the model or identify whether there is extra information. These rules are called existential rules and they do not offer any alternative solution other than marking the location of the problem.

As it was previously explained, each rule may require other rules to be evaluated successfully before they can be checked. Figure 6 presents the dependencies between the validation rules that are currently enforced by the engine. These dependencies are established between rules of the same or of different contexts.

In the Figure, the identifier of each rule is divided in three parts. The first part indicates the context to which the rule belongs (Model, Element - EC, Attribute - AC, Contain-

---

**Table 2: Linguistic problems**

<table>
<thead>
<tr>
<th>RULE</th>
<th>Problem detected</th>
</tr>
</thead>
<tbody>
<tr>
<td>MC-LR-1</td>
<td>The value of MetamodelURI is not provided</td>
</tr>
<tr>
<td>MC-LR-2</td>
<td>No instance matches the root EClass in the domain MM</td>
</tr>
<tr>
<td>EC-LR-1</td>
<td>An Element instance has not value in typeName</td>
</tr>
<tr>
<td>EC-LR-2</td>
<td>The value of typeName in one Element instance has blanks</td>
</tr>
<tr>
<td>EC-LR-3</td>
<td>The Element instance does not have an owner instance</td>
</tr>
<tr>
<td>AC-LR-1</td>
<td>The Attribute instance does not have any value in typeName</td>
</tr>
<tr>
<td>AC-LR-2</td>
<td>The typeName in one Attribute instance has blanks</td>
</tr>
<tr>
<td>AC-LR-3</td>
<td>The typeName in one Attribute instance starts or ends with comma</td>
</tr>
<tr>
<td>AC-LR-4</td>
<td>There are several Attribute instances with the same value typeName belongs to the same Element instance</td>
</tr>
<tr>
<td>AC-LR-5</td>
<td>The value of one Attribute instance does not correspond with the type</td>
</tr>
<tr>
<td>CRC-LR-1</td>
<td>The ContainmentRelation instance does not have any value in typeName</td>
</tr>
<tr>
<td>CRC-LR-2</td>
<td>The typeName in one ContainmentRelation instance has blanks</td>
</tr>
<tr>
<td>CRC-LR-3</td>
<td>Two ContainmentRelation instances associate two Element instances with opposite direction</td>
</tr>
<tr>
<td>CrRC-LR-1</td>
<td>The CrossRelation instance does not have any value in typeName</td>
</tr>
<tr>
<td>CrRC-LR-2</td>
<td>The typeName in one CrossRelation instance has blanks</td>
</tr>
</tbody>
</table>
Table 3: Ontological problems

<table>
<thead>
<tr>
<th>RULE</th>
<th>Problem detected</th>
</tr>
</thead>
<tbody>
<tr>
<td>MC-OR-1</td>
<td>The domain MM has several root EClasses</td>
</tr>
<tr>
<td>MC-OR-2</td>
<td>The value of the attribute MetamodelURI does not coincide with domain MM URI</td>
</tr>
<tr>
<td>EC-OR-1</td>
<td>The typeName of one Element instance does not match with any EClass name</td>
</tr>
<tr>
<td>EC-OR-2</td>
<td>The Element instance has several owner instances</td>
</tr>
<tr>
<td>EC-OR-3</td>
<td>There are several Element instances that match with the root EClass in domain MM</td>
</tr>
<tr>
<td>EC-OR-4</td>
<td>The Element instance does not have associated Attribute instances required</td>
</tr>
<tr>
<td>EC-OR-5</td>
<td>The Element instance does not have associated the ContainmentRelation instances required</td>
</tr>
<tr>
<td>EC-OR-6</td>
<td>The Element instance does not have associated the CrossRelation instances required</td>
</tr>
<tr>
<td>AC-OR-1</td>
<td>The typeName of one Attribute instance does not match with any EAttribute name of the corresponding EClass</td>
</tr>
<tr>
<td>AC-OR-2</td>
<td>The type in one Attribute instance does not match with the corresponding EType</td>
</tr>
<tr>
<td>AC-OR-3</td>
<td>The attribute value in one Attribute instance is not defined</td>
</tr>
<tr>
<td>AC-OR-4</td>
<td>The quantity of values in the attribute value for one Attribute instance is lower than the lower bound in the domain MM</td>
</tr>
<tr>
<td>CRC-OR-1</td>
<td>The typeName in one ContainmentRelation instance does not match with any EReference name of the corresponding EClass</td>
</tr>
<tr>
<td>CRC-OR-2</td>
<td>The quantity of ContainmentRelation instances belong to one Element instance is lower than the lower bound in the domain MM or greater than the upper bound in the domain MM</td>
</tr>
<tr>
<td>CRC-OR-3</td>
<td>The typeName in one CrossRelation instance does not match with any EReference name of the corresponding EClass</td>
</tr>
<tr>
<td>CRC-OR-4</td>
<td>The quantity of CrossRelation instances belong to one Element instance is lower than the lower bound in the domain MM or greater than the upper bound in the domain MM</td>
</tr>
</tbody>
</table>

The second part indicates whether the rule is linguistic (LR) or ontological (OR); and the last part is a rule consecutive. As an example, according to Figure 6 the rule AC-OR-1 requires the previous validation of rule EC-OR-1: AC-OR-1 validates whether the typeName of one Attribute instance matches with some EAttribute name of the corresponding EClass in the domain metamodel; however, this can only be validated after EC-OR-1 asserted that the attribute typeName of the Entity matches the name of some meta-type in the domain metamodel.

3.4 User assistance

There are two types of assistance that GraCoT provides to users. The first type is made of 8 wizards that serve to create diagrams, select a new domain metamodel, export a model to xmi, validate a model, and establish the GraCoT configuration.

The second type of assistance guides the user during the co-creation of the model and the domain metamodel. This type of assistance is available to fix the problems classified as warnings by the RV. Therefore, the solutions provided by the relevant validation rules are shown as quickfix options. After the user selects one of the suggested solutions, the inconsistency may or may not be solved immediately, depending on whether additional information is required. If this occurs, GraCoT requests the necessary information to continue with the process. GraCoT currently offers 13 wizards of this second type.

Tables 4 and 5 show that the suggested alternative solutions provide for each validation rule, making conformance inconsistencies as warnings. For each suggested solution it is also stated whether it is interactive (requires additional information) or not. In Table 4, all the solutions except for the one related to rule AC-LR-5 apply the changes over the model. In Table 4, the last column indicates whether the change applies to the model (M) or to the domain metamodel (MM).

3.5 Standard formats

In order to support the compatibility with the EMF framework and with other modeling tools, GraCoT offers services to import and export xmi models. In the first case, the imported model must conform to the domain metamodel in order to be converted into a GIMM model. This is achieved by means of a transformation embedded in GraCoT that produces the GIMM model and sets the proper metamodelURI attribute.

In the second case, GraCoT transforms a GIMM model into an EMF model, and then stores it using the xmi format. This export operation can only be applied if the GIMM model has been validated successfully, and results in a model that conforms linguistically and ontologically to the domain metamodel.
Table 4: Suggested solutions to validation rules

<table>
<thead>
<tr>
<th>Rule</th>
<th>Suggested solution</th>
<th>Interactive?</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>MC-LR-1</td>
<td>Set metamodelURI with current URI</td>
<td>No</td>
<td>M</td>
</tr>
<tr>
<td>EC-LR-1</td>
<td>Set an automatic Element typeName</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>RC-OR-1</td>
<td>Remove blanks</td>
<td>No</td>
<td>MM</td>
</tr>
<tr>
<td>AC-LR-2</td>
<td>Set an automatic Attribute typeName</td>
<td>No</td>
<td>MM</td>
</tr>
<tr>
<td>AC-LR-3</td>
<td>Remove blanks</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>CRC-LR-1</td>
<td>Set an automatic ContainmentRelation typeName</td>
<td>No</td>
<td>MM</td>
</tr>
<tr>
<td>CRC-LR-2</td>
<td>Remove blanks</td>
<td>No</td>
<td></td>
</tr>
</tbody>
</table>

Table 5: Suggested solutions to ontological validation rules

<table>
<thead>
<tr>
<th>Rule</th>
<th>Suggested solution</th>
<th>Interactive?</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>MC-OR-2</td>
<td>Update metamodelURI with current URI</td>
<td>No</td>
<td>M</td>
</tr>
<tr>
<td>EC-LR-1</td>
<td>Select a valid Element typeName</td>
<td>Yes</td>
<td>M</td>
</tr>
<tr>
<td>AC-OR-1</td>
<td>Add EClass</td>
<td>Yes</td>
<td>MM</td>
</tr>
<tr>
<td>AC-OR-2</td>
<td>Set the valid Attribute type</td>
<td>Yes</td>
<td>MM</td>
</tr>
<tr>
<td>AC-OR-3</td>
<td>Use a default value</td>
<td>No</td>
<td>M</td>
</tr>
<tr>
<td>AC-OR-4</td>
<td>Add and remove values</td>
<td>Yes</td>
<td>M</td>
</tr>
<tr>
<td>CRC-OR-1</td>
<td>Add a valid ContainmentRelation typeName</td>
<td>Yes</td>
<td>MM</td>
</tr>
<tr>
<td>CRC-OR-2</td>
<td>Set current cardinality</td>
<td>No</td>
<td>MM</td>
</tr>
<tr>
<td>CRC-OR-3</td>
<td>Modify cardinality</td>
<td>No</td>
<td>MM</td>
</tr>
<tr>
<td>CRC-OR-4</td>
<td>Add Cross EReference</td>
<td>Yes</td>
<td>MM</td>
</tr>
<tr>
<td>CRC-OR-5</td>
<td>Set current cardinality</td>
<td>No</td>
<td>MM</td>
</tr>
</tbody>
</table>

Figure 7: Domain metamodel and GIMM model

The second problem is marked over two instances of the relation “clients”, which links instances of Element with type Name “Enterprise” and “Client”. The validation rule associated with this problem has id CRC-OR-2. This rule is also an ontological validation rule, and it checks the cardinality of relations. In this case, there is a problem because the cardinality in the metamodel is 1, but in the model there are two instances of the relation.

An additional detail to consider is that the validation of the first problem depends on the successful validation of rules AC-LR-2, AC-LR-3 and EC-OR-1, which validate whether the typeName of the Attribute instance is well-formed, and the typeName of the corresponding Element instance matches with some EClass in the domain metamodel. On the other hand, the second problem is validated if rule CRC-OR-1 was previously checked. CRC-OR-1 validates that the typeName in the ContainmentRelation instance matches with some EReference name of the corresponding source and target EClass in the domain metamodel.

For each of the aforementioned problems, a quickfix with the suggestions listed in table 5 is shown. These are complemented by a suggestion to ignore the warning and keep the model in a non conformity state. Figure 8 shows the quickfix that offers alternative solutions to the second problem.

For the first problem (wrong type in the attribute “code”) the solution is not interactive because it does not need additional information from the user. To recover the conformance and fix the problem, it is only necessary to change the type of the Attribute to the EType specified in the domain metamodel.

On the other hand, in the case of the second problem the intervention of the user is necessary. If the second offered
Figure 8: Quickfix for an invalid Element instance

Figure 9: The wizard for updating a cardinality

suggestion is selected (’Modify the cardinality of “clients” in the domain metamodel’), the user needs to provide the new cardinality for the relation. The wizard presented to the user is shown in Figure 8. After the wizard is completely executed, the domain metamodel is modified and the problem in the model is resolved. Also at this point, some validation rules are regenerated in order to verify the conformity of the model against the updated domain metamodel.

Figure 10: A GIMM model exported

After each problem is solved, new problems may appear because of the dependencies between rules. For instance, another way of solving the first problem is by changing the type of the Attribute to “EBoolean” both in the model and the domain metamodel. In this case, the validation engine would then check the rule AC-LR-5 and detect that the provided value (26) is not assignable to the type “EBoolean”.

Finally, when the model conforms to the domain metamodel in a linguistic and ontological way, it is possible to export it to an EMF model in xmi format. Figure 10 illustrates the full model from Figure 5 (in the GraCoT canvas), and also the equivalent EMF model. In the Figure, some corresponding points are marked using arrows.

5. RELATED WORK

In this Section we briefly present some previous works that are somehow related to our own. In the first place, the EMF framework provides Dynamic EMF framework to create models and metamodels using a programmatic interface. However, in contrast to our work, it handles models and metamodels through a strong conformance relation. This condition makes it complicated to change or replace metamodels at runtime, without constant application of transformations to the models under construction.

In [11], Gabrysiak et al. discuss how metamodels can be used in a flexible way. Although this work does not propose any tool, it is very interesting because they present a classification of approaches based on the dynamicity of the metamodels in the tools. This classification is separated in three categories: before modeling, modeling captured insights and after modeling. Consequently, it is possible to assert that GraCoT belongs simultaneously to the first two. With respect to the first category, GraCoT supports user-generated metamodels; that is, metamodels are designed by the users of the tools and not by the developers of the tools. With respect to the second, GraCoT provides support for the co-creation of models and metamodels, and also for the co-evolution of these two aspects.

In [17], Ubayashi et al. propose a reflective editor for the construction of models, and a strategy with similar goals related to our work; particularly, they also co-create models and metamodels. However, there are fundamental differences in the approaches. Firstly, their approach is specifically targeted to aspect-oriented modeling, where the changes introduced in metamodels are extensions to model additional aspects. Secondly, when the metamodel changes, the editor is regenerated; in contrast with GraCoT which is never regenerated.

The Reflective Ecore Model Diagram Editor [9] is a graphical editor based on GMF to manipulate EMF models independently of a metamodel like GraCoT. This editor is capable of dynamically loading a metamodel, and of creating models conforming to it, but it had some restrictions related to the way it handled relations and attributes from the metamodel. In addition, it offered a dynamically generated tool palette with the element types obtained from the metamodel. However, this tool is not capable of identifying conformance problems and resolves them dynamically. Unfortunately, the project has been abandoned since 2009 and is just compatible with the Eclipse, EMF and GMF versions of today.
In [15], Sanchez et al. propose an interesting tool for Eclipse to create a metamodel from model fragments constructed by end-users with tools with sketching facilities, such as Dia. These fragments are translated into textual fragment that it will be annotated by engineers in order to assign meaning. Next, the metamodel is updated according to this textual information, and the new version of the metamodel will have the corresponding annotations. The changes in the metamodel that do not break the conformance are applied without asking the user. But when the changes are unresolvable, they are classified as automatic, a conflict or as a suggestion. When a change is classified as automatic, the tool takes one decision of several available options and informs the user about the decision taken. When a change is classified as a suggestion, the user must take a decision of the solution option to be applied. Although this editor is capable of creating the metamodel, it does not offer the option to modify it directly, and only the visualization of the metamodel is possible. Finally, the metamodel can be exported to different target platforms, such as EMF and MetaDepth.

6. CONCLUSIONS

In this paper we have discussed some needs of the modeling and metamodeling processes in specific domains; particularly, the need to define the metamodel in advance to the creation of the model in the Enterprise Architecture context. In addition, we discuss about the lack of dynamicity of current model editors, such as EMF, to support the manipulation of models and metamodels at runtime.

In order to alleviate this situation, we present GraCoT (Graphical Co-creation Tool) as a graphical tool capable to support the co-creation of models and metamodels in an incremental and simultaneous way. In addition, the strategy behind GraCoT is described, which is based on the separation of the linguistic and ontological aspects.

GraCoT is presented as a fully functional tool. This paper presented a description of the architecture proposed, which is based on Eclipse technologies, details specifications of the validation engine that detects the conformity problems in the models, the user support which guide users in order to recover the conformance, and the manipulation of the standard xmi format.

To illustrate the capabilities of GraCoT, the paper presented an example that covers it main features. This example presents the co-creation process to manipulate and solve the conformance problems of one small model and its corresponding metamodel.

7. REFERENCES
