Coevolution Assistance for Enterprise Architecture Models

Hector Florez
Universidad de los Andes
Department of Systems and Computing
Engineering
Bogotá, Colombia
ha.florez39@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

Germán Vega
Université de Grenoble
Equipe ADELE
Grenoble, France
german.vega@imag.fr

ABSTRACT
When metamodels evolve, model conformity may be broken. This forces the owners of the models (modelers) to intervene because it is impossible to automatically discover what to change in order to regain conformity. This paper presents ASIMOV, a platform for model and metamodel co-evolution based on two hypothesis: i) a metamodeler knows the rationale behind metamodel changes, and is capable of providing guidelines for model coevolution; ii) the modeler is the only one in grade of making final decisions about his models. ASIMOV provides two languages for metamodelers: ASIMOV Evolution, to specify changes in the metamodels; and ASIMOV Assistance, to propose corresponding changes in the models. Also, ASIMOV Engine solves automatically the changes in models that can be automatically solved and assists modelers in coevolving their models to regain conformity. Moreover, modelers can adapt the proposed changes to suit their particular needs, introducing additional information when it is required. ASIMOV is here illustrated in the context of Enterprise Architecture projects.

Keywords
Model coevolution, enterprise architecture modeling, modeler assistance, model driven engineering

1. INTRODUCTION
A metamodel is the abstraction of the concepts of a specific domain of information, and it is usually built by one metamodeler, who is the person that knows the domain, when he/she solves a problem using an MDE approach. On the other hand, one model is a representation of a specific case of a domain, and conforms to the metamodel of the domain. If the domain changes, the metamodel needs to evolve; as a consequence, each model that conforms to the metamodel that evolved, needs to be readjusted in order to maintain the conformity. The coevolution refers to the process applied to one model that conforms to one metamodel, due to changes done in the metamodel. However, there are some cases where models do not need to coevolve due a change in the metamodel. For instance, the change in the metamodel is the addition of a new concept, or there are not instances conform to the changed concept. The changes in the metamodel are given by metamodelers, yet modelers apply the coevolution on the model.

In addition, in Enterprise Architecture (EA) context, the fundamental goal is the abstraction between the organization and the Information Technologies (IT). This abstraction is represented by a model which contains the business and IT elements, and conforms to a metamodel created for the organization. Moreover, in this context there are just one metamodel in one organization; also, this metamodel has a large number of elements, has great complexity, and usually is built by one person.

Unfortunately, in several cases it is not possible to automatically modify the models to make them conform to the updated metamodels. Although a few kinds of changes over metamodels result in changes to the models that can be automatically calculated, in several cases human intervention is required [1, 3]. This is so because certain changes over metamodels require introducing additional information into the conformant model, the reorganization of the information already present, or even the deletion of some parts. Modelers (persons that own, build, understand, and use the models and its information) are the only ones in grade of making proper decisions about its evolution. Failing to include them in this process may result in incomplete, inconsistent, or incorrect models, which may conform to the metamodels, yet can be inappropriate from the point of view of the domain.

There are several techniques to manage metamodel and model coevolution. These techniques typically decompose the problem into two smaller ones [4]: 1) identifying the changes to the metamodels, and 2) modifying the models to regain conformity with corresponding metamodel. Said coevolution techniques have been classified by the strategies that they use to solve each problematic [3]. However, for the EA context, the technique must include modeler intervention.

This proposal presents a coevolution solution strategy for the EA context, where we assume that a metamodel is usually built by one metamodeler, and a conformant model is usually created by one modeler. In addition, the metamodel
evolution takes place in the event that metamodelers, who know the domain, define the changes over the metamodel, and add alternatives to solve the changes on the model that are not automatically resolvable. One modeler can apply the coevolution specification to a specific model that conforms to the changed metamodel by selecting among the alternatives defined by metamodelers. This selection applies for each instance related with the changed elements in the metamodel whose coevolution has been specified. Furthermore, this approach avoids inconsistencies introduced by modeler decisions due they must follow certain rules established by metamodelers. Additionally, in this proposal, coevolution can be executed step by step allowing modelers to store the process, enabling restarting it in a specific point. Moreover, it allows rolling back any instruction in case of modelers change decisions about the coevolution of a specific model.

The rest of the paper is structured as follows. Section 2 describes the model coevolution problem and presents some solutions proposed in the past. Section 3 presents our proposed strategy for solving the model coevolution problem. In section 4, we present the two languages developed to solve evolution of the metamodels and the correspondent coevolution of the models. In section 5, we present the proposal evaluation. Finally, section 5 presents the conclusions.

2. MODEL COEVOLUTION
2.1 The coevolution problem
The evolution is a common event in the life cycle of a metamodel. This phenomenon happens when the models of the information are created by humans [2]. The main reason why metamodels evolve is that the metamodel may be incomplete. In this case the evolution of a metamodel is driven by the need of fixing it to become more complete [11]. In EA context, these changes occur very often due the construction of the metamodels and models is an iterative process.

Changes on one metamodel impact all models that conform to it. The problem is presented when an element from the metamodel changes, and the model does not change breaking the conformity of the dependent model. Figure 1, illustrates the coevolution problem. After evolution $\Delta$ of a metamodel $MM$ into $MM'$, the goal is to coevolve $m$ that conforms to $MM$, to $m'$ that conforms to $MM'$, by creating an appropriate coevolution migration $M$ [12].

![Figure 1: The coevolution illustrated.](image)

2.2 Coevolution solutions
There are some approaches that solve this problem focusing in two aspects: identifying the differences between the original and the evolved metamodels, and making the modifications on the model in order to be conformant to the new metamodel [3]. The first approach uses a declarative evolution specification to define a difference metamodel which can be calculated from identified changes in the metamodel [3]. The second approach specifies metamodel evolution by a sequence of operations where each operation is applied on metamodel and model level [1, 10, 5].

This paper attempts to solve the coevolution making modifications on the model in order to be conformant to the new metamodel. Common approaches establish that the possible changes in metamodels can be classified as 1) non-breaking (NB), which are the changes that have no impact on the model; 2) breaking and automatically resolvable (BAR), which are the changes that have impact on the model, yet can be resolved automatically; and 3) breaking and not automatically resolvable (BNAR), which are the changes that cannot be inferred, so need additional information that is provided by the modeler in order to be fulfilled [1, 4, 12, 13, 14, 7, 15]. Having identified the changes each approach resolves the first two categories of changes automatically using different frameworks for model transformation. To address the last category, some approaches take advantage of the user assistance to coevolve the models [1].

Becker et al. [1] propose an approach to address BNAR changes through a framework for assisting the user in the definition of model coevolution when a change of this category is found.

In Cicchetti et al. [3], the proposal classifies the changes in atomic changes and define the process of coevolution. Then, create a differential metamodel with the identified changes, and it is classified in two new metamodels, the ones that are BAR and the ones that are BNAR. If there are no relations between the two metamodels, each metamodel is executed separately. If there are relations between the two metamodels, the coevolution is done stepwise using user intervention.

In Herrmannsdorfer et al. [8], approach the coevolution through the proposal called COPE that is a language to satisfy two requirements: 1) reuse of recurring migration knowledge and 2) expressiveness to support domain specific migrations. COPE allows to create coupled transactions that are a combination of metamodel adaptation and model migration. COPE resolves the coevolution executing those coupled transactions created by one metamodeler and its execution does not require user intervention.

In Garces et al. [4], approach the coevolution through ATG (Adaptation Transformation Generation) that is a semi-automatic approach to generate an executable model adaptation transformation generating the adaptation transformation in two steps: 1) retrieving the relationships between the elements of the metamodel’s versions identifying the differences between the metamodels, and 2) taking these relationships as inputs to generate a High Order Transformation that will generate the required transformation to coevolve the model.

In Gruschko et al. [6], propose to coevolve the model using a set of automatic transformations defined previously solving the problems in one of the next three categories: addition, delete or rename. When a BNAR change is found, the user should specify the way that the elements are going to change by creating a set of transformation rules using ETL.
3. THE ASIMOV STRATEGY

Our proposal called ASIMOV consists of a strategy where the domain experts that modify the metamodels explicitly specify solution alternatives for BNAR changes. ASIMOV identifies two different roles: metamodelers and modelers. Metamodelers, who are domain experts in the EA context and know the reasons why a metamodel needs to evolve, define the set of changes over one metamodel, and different possible solutions to apply on the conformant models for BNAR changes. Modelers use the evolution and co-evolution script made by metamodelers to apply it over a specific model, and he/she decides how the model changes based on the possible solutions described by metamodelers. The metamodeler writes the changes over the metamodel in an evolution script using the ASIMOV Evolution Language (See 4.1); additionally, the metamodeler defines in the same script some assistance specifications for each BNAR change using the ASIMOV Assistance Language (See 4.2).

The ASIMOV Engine assists modelers in solving the BNAR changes in a stepwise approach, presenting the coevolution solutions alternatives proposed by metamodelers. One modeler selects among the different alternatives for each affected instance, and provides any additional information required by the selected alternative. For instance, if the metamodel evolution proposed by the metamodeler includes the addition of a new attribute, solution alternatives can include the initialization of this attribute with a predefined value or with a derived value based on other already existing attributes. If these alternative solutions are defined when a specific model must coevolve, the modeler must select which of these solutions to use. Additionally, ASIMOV Engine allows modelers to provide additional information to complete the change on the model if necessary. For instance, if the new attribute must be initialized, the engine must also request the modeler for the initial value.

ASIMOV achieves the coevolution based on independent migration transformations, where each one of them is related with one evolution instruction applied on the metamodel. Each migration transformation affects just the instances related with the element changed in the metamodel. As a result, each independent migration transformation generates an intermediate metamodel and conformant model. Figure 2 shows the coevolution process.

Figure 2: ASIMOV Transformation Strategy.

In this stepwise approach, the metamodel \( MM \) evolves to the metamodel \( MM' \) through \( \Delta \) that is a set of transformations \( \delta_i \) that create intermediate metamodels \( MM_i \). Also, the model \( m \) that conforms to \( MM \) is migrated to \( m' \) that conforms to \( MM' \), through the migration transformation \( M \) that is a set of migrations \( \mu_i \) that creates intermediate models \( m_i \). Each intermediate model \( m_i \) conforms to the intermediate metamodel \( MM_i \).

In a single step, for each \( \delta_i \), a \( \mu_i \) is applied based on the selection made by the modeler, who is performing the coevolution, over a specified model \( m \). The migration transformation \( \mu_i \) is performed based on \( A_{\delta_i} \). Then, the intermediate model \( m_i \) is generated based on \( m_{i-1}, \mu_i \), and \( A_{\delta_i} \).

\( A_{\delta_i} \) is made up of a set of assistance options \( \{a_{\delta_1,1}, a_{\delta_1,2}, \ldots, a_{\delta_1,n}\} \) written by metamodelers for each BNAR change. \( \Gamma_{\delta_i} \) is the set of the instances in the model \( \{\gamma_{\delta_1,1}, \gamma_{\delta_1,2}, \ldots, \gamma_{\delta_1,n}\} \) related with the change in the metamodel \( \delta_i \). One modeler performs the coevolution for a specific model \( m \), selecting for each transformation \( \delta_i \) an assistance option \( a_{\delta_1,j} \), for each instance \( \gamma_{\delta_1,k} \). The ASIMOV algorithm is presented in Algorithm 1.

Algorithm 1

```
for all \( \delta \) in \( \Delta \) do
  if \( \delta \) is BNAR then
    for all \( \gamma \) in \( \Gamma_{\delta} \) do \( \triangleright \Gamma_{\delta} \)-instances related with \( \delta \)
      \( a \leftarrow \text{selectOneFrom} \ A_{\delta} \)
      if \( a \) requires data then
        data \( \leftarrow \text{request data} \)
        apply\((\gamma,a,\text{data})\)
      else
        apply\((\gamma,a)\)
    end if
  end for
else
  for all \( \gamma \) in \( \Gamma_{\delta} \) do
    apply\((\gamma)\)
  end for
end if
```

In addition, ASIMOV allows applying a specific assistance option \( a_{\delta,1,j} \) to several instances. Consequently, the model \( m_i \) contains the updated instances related with the correspondent change in the metamodel depending the modeler decisions. Figure 3 shows a single step with assistance options in the coevolution process.

Figure 3: Single step assistance options.

4. ASIMOV LANGUAGES AND TOOL SUPPORT

ASIMOV resolves the evolution and coevolution of metamodels and models respectively, by defining two languages. The first one, called ASIMOV Evolution Language used to describe the metamodel evolution \( \Delta \); and the second one, called ASIMOV Assistance Language is used to define how to resolve those changes \( A_i \) in the model to guarantee conformity with the evolved metamodel in case of those changes.
are BNAR. To address these cases, the ASIMOV Assistant Language is included in the corresponding instructions of the ASIMOV Evolution Language.

### 4.1 ASIMOV Evolution

ASIMOV Evolution is a language based on a catalog of possible changes that one metamodel can present. It is planned to allow metamodelers to write programs easily. This language permits generating an output metamodel based on an input metamodel where the output metamodel includes the changes described by the evolution script. Additionally, when the output metamodel includes NB or BAR changes, the language applies the correspondent changes over the model to maintain conformity with the output metamodel.

The structure of ASIMOV Evolution Language consists in a basic building block that is the evolution clause. This clause encapsulates all instructions in the program and requires indicating the input metamodel, the output metamodel, the input model, and the output model using the clauses fromMM, toMM, fromM, and toM respectively. After including metamodels and models, metamodelers can write the instructions corresponding to the evolution. These instructions are equivalent with $\delta$. Each instruction must be inside a block identified with an instruction clause. This block includes the evolution instruction, and comments for this instruction. Through these comments, metamodelers justify the changes, for modelers understand them.

A complete instruction catalog comprehends a set of operations which can be classified according to several properties that allow evolving any source metamodel to any target metamodel [10]. To achieve this, our proposal is completeness from the principle that each instruction has high granularity, which implies that the operation cannot be decomposed into smaller operations [9], to ensure unitary changes on the metamodel in the evolution process.

Hence, ASIMOV Evolution Language has a catalog that is made up of 24 instructions classified in NB changes, BAR changes, and BNAR changes. With these instructions metamodelers can make the necessary changes on the classes, attributes and references from the input metamodel. Table 1 presents the instruction catalog created for ASIMOV Evolution Language. With this instruction catalog, ASIMOV Evolution Language offers a language that supports a great variety of metamodel evolution cases.

<table>
<thead>
<tr>
<th>Change Type</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-breaking (NB)</td>
<td>setNonAbstractClass decreaseAttributeLowerBoundTo increaseAttributeUpperBoundTo addReference decreaseReferenceUpperBoundTo</td>
</tr>
<tr>
<td>Breaking and automatically resolvable (BAR)</td>
<td>renameClass renameAttribute renameReference</td>
</tr>
<tr>
<td>Breaking and not automatically resolvable (BNAR)</td>
<td>deleteClass setAbstractClass newSuperClass deleteSuperClass addAttribute deleteAttribute increaseAttributeLowerBoundTo decreaseAttributeUpperBoundTo modifyTypeOfAttribute deleteReference increaseReferenceLowerBoundTo decreaseReferenceUpperBoundTo changeReferenceSource changeReferenceTarget</td>
</tr>
</tbody>
</table>

### 4.2 ASIMOV Assistance

The ASIMOV Assistance Language consists of a set of instructions that allow metamodelers to specify the way that changes are applied to one conformant model for the BNAR changes. Taking advantage of the granularity level of ASIMOV Evolution, the assistance clause can be placed inside the instruction block. The assistance block must be defined after the instruction referring to the change in the metamodel. For each instruction, one metamodeler can include several assistance blocks, giving modelers the opportunity of selecting for each instance of the concept changed, a different assistance instruction from the given options. In addition, the assistance process is sequential, so the order of the instructions and assistance is very important in order to guarantee the semantics in the model.

The grammar of ASIMOV Assistance Language is a subset of Java grammar whose is possible to allowing resolving: variables declarations, arithmetic operations, compare operations, concatenate operations, logical operations, iteration functions, condition functions, input functions, and output functions. As a result, this subset of Java grammar offers several advantages: 1) the grammar is well known by metamodelers with some experience in Java, 2) the assistance scripts are flexible, 3) the assistance scripts can use services implemented in libraries written in full Java grammar, and 4) metamodelers can create libraries with reusable assistance scripts to solve assistance patterns.

Variable declarations allow creating in dynamic memory values with any type supported by Java. Also, these declarations can be written for all instructions, one instruction, or a specific assistance for a specific instruction. In another hand, each instruction classified as BNAR, necessarily has parameters that reference objects corresponding to elements involved in the instruction. For instance, the instruction `addAttribute` requires the attribute to add, the class where the attribute will be added, the type of the attribute, and the instances of the attribute in the model.

In addition, ASIMOV Assistance Language allows including libraries written in Java in order to reduce complexity in the scripts, giving to metamodelers another mechanism to write coevolution scripts. However, it is necessary to import the required libraries using the `import` clause at the beginning of the script. Metamodelers can develop their own libraries and include them in a specific coevolution or can use predefined libraries available in ASIMOV that solve specific operations over instances in the model. These libraries contain several Java classes with methods that allow typical operations such
Listing 1: Coevolution example
1 import library.attributeServices.AddAttribute;
2 evolution myEvolutionProgram{
3 fromMM: "./models/sourceMM.ecore"
4 toMM: "targetMM.ecore" "targetURI"
5 fromM:. ./models/sourceM.ecore"
6 toM: "targetM.xmi"
7 instruction{
8 addAttribute name{
9 className=Product
10 type=String
11 }
12 [EAttributeImpl theAttribute, 
13 EClassImpl theClass, 
14 EDataTypeImpl theType, 
15 ArrayList modelClassInstances]
16 comments{
17 "Addition of product name for its identification"
18 }
19 assistance{
20 comments{
21 "Initiate the attribute with default value" 
22 }
23 AddAttribute.initializeStringValue(
24 theAttribute, modelClassInstances);
25 }
26 assistance{
27 comments{
28 "Initiate the attribute with input String value" 
29 }
30 AddAttribute.initializeStringValue(
31 theAttribute, modelClassInstances);
32 }
33 }
34 }

The modeler, who performs the coevolution can select the assistance and can select several instances through a graphic user interface for each change in the metamodel. Figure 4 presents the GUI for the assistance selection presenting: 1) the instruction δ; comments (line 16) in the title; 2) the information about δ; (lines 7, 8) below the title; 3) the set of instances Γδ; and 4) the assistance Aδ; comments (lines 20, 27). The GUI requires that the modeler selects assistance for all instances. Once it is done, the modeler can make the assistance selections for the instances related with the next change in the metamodel.

4.3 ASIMOV Engine

Once a modeler decides to coevolve a model, the ASIMOV engine must be used to perform that task. The ASIMOV Engine receives the evolution and coevolution script, the affected metamodel, and a conformant model. Then, the engine starts the process applying the changes in the same order than the instructions have been defined by the meta-modeler. During runtime, the ASIMOV Engine creates an intermediate metamodel and an intermediate model for each instruction executed in order to keep the conformity at all times. This feature of ASIMOV allows metamodelers to reference the current and previous model independently. This feature ensures that the model does not lose information during the coevolution process, and provides the possibility of storing the process of any step or position as well as restoring the process. For debugging purposes, the script can be executed in a step-by-step basis. Each script can be also executed not completely, but from the beginning to a specific instruction or starting in a specific instruction to the end. In all cases, each coevolution instruction must be applied to all affected elements in the model, and not to a subset of the affected elements. For example, if the evolution script modifies an EClass in the metamodel, all the model elements that conform to that EClass are also coevolved.

5. EVALUATION

In order to evaluate the proposal\(^1\), we have selected a real scenario, which is a commercial scenario named Alps Furniture (AF) built in the Enterprise Architecture Laboratory at Universidad de los Andes. This scenario has a complex metamodel; however, for this evaluation we used one specific domain named BusinessPartners (BP). The objective of this evaluation is to demonstrate that 1) ASIMOV is useful for metamodelers and modelers that are working in the EA context, and 2) ASIMOV has a great level of expressiveness and usability. To achive this, we made two experiments.

The first experiment consists in the coevolution process using two models conform to the AF BP metamodel. The coevolution has been performed by several modelers using EMF editing tools and ASIMOV. Also, we measured to modelers the next variables: time, number of actions, errors, and attempts to coevolve the models. Based on the results, the time used by modelers to coevolve a model using ASIMOV is around 19% of the time required to coevolve the model using EMF tools. Also, due ASIMOV can apply a specific assistance to several instances, the number of actions ap-

---
\(^1\)See: backusi.uniodes.edu.co/~enar/dokuwiki/doku.php?id=asimovevaluation
plied to a model using EMF is almost three times than the number of actions using ASIMOV. In addition, ASIMOV is intuitive; as a result, the number of error trends to zero; in contrast, it is easy to place several errors changing the model using EMF. Finally, due the errors, the number of attempts is higher using EMF.

The second experiment consists in the coevolution process using a third model conforms to the AF metamodel in the BP domain. This third model has much greater number of instances. Based on the results of the previous experiment and this new experiment, we measured average time and average number of actions. We compare these variables considering the number of instances of each model. Based on this second experiment, the average increment in time having three models using EMF is 85% and using ASIMOV y 16%; and the average increment in actions using EMF is 126% and using ASIMOV is 23%.

As a result, we could prove that ASIMOV 1) performs the coevolution in EA context efficiently and effectively; 2) offers to modelers knowledge about metamodel evolution through the information written by metamodelers; 3) can solve the coevolution in a complex problem; 4) ensures model conformity in all steps of the process; and 5) is usable for EA metamodelers and modelers, for it’s languages are clear and easy to use by metamodelers and it’s GUI is intuitive for modelers when they apply the coevolution selecting the alternatives for each instance in the model for a related BNAR change.

6. CONCLUSION
A coevolution process, where metamodelers can communicate the solution for the changes that are BNAR, is possible. In our approach, two DSLs allow EA metamodelers not only to define the changes to the metamodel, but also the way to deal with changes if a specific action must be taken by modelers when the coevolution process is applied.

An advantage of this approach is that modelers cannot perform illogical changes on the model to make it conformant to an evolved metamodel when BNAR changes occurred. Modelers must select an alternative from a set of logical defined changes provided by metamodelers. Another advantage of this approach is, based on the execution of the coevolution as a set of atomic modifications over the metamodel, each transition of the model happens after correspondent transition on the metamodel. Thanks to this, model conformity can be ensured as the coevolution is taking place.

ASIMOV not only improves the time and number of actions required by modelers in the coevolution process, but ensures the model conformity after each change over the correspondent metamodel. In addition, with ASIMOV the coevolution in models with a large number of instances is possible with a simple intervention by the modeler. As a result, ASIMOV solves properly the coevolution in EA context due it is applicable considering EA modeling requirements.

7. REFERENCES