A NATURAL LANGUAGE REQUIREMENTS ENGINEERING APPROACH FOR MDA

María Carmen Leonardi, Marcela Ridao, María Virginia Mauco, Laura Felice[1]
[1]INTIA; UNCPBA; Argentina

ABSTRACT

A software system for any information system can be developed following a model driven paradigm, in particular MDA (Model Driven Architecture). In this way, models that represent the organizational work are used to produce models that represent the information system. Current software development methods are starting to provide guidelines for the construction of conceptual models, taking as input requirements models. In MDA the CIM (Computation Independent Model) can be used to define the business process model. Though a complete automatic construction of the CIM is not possible, we have proposed in other papers the integration of some natural language requirements models and we have defined a strategy to derive a CIM from these models. In this paper, we present an improved version of our ATL transformation that implements a strategy to obtain a UML class diagram representing a preliminary CIM from requirements models allowing traceability between the source and the target models.

KEYWORDS

ATL transformations, MDA, Natural Language Requirements Models, Traceability

1. INTRODUCTION

A software system is always embedded in an environment; therefore, its requirements can not be defined without a proper consideration of the environment in which the system is embedded [21]. Definition of the system context, also called Universe of Discourse (UofD), may be considered the first phase in a software development process, and the models generated during this phase serve as an input for later phases, such as conceptual modeling [8]. The participation of stakeholders is crucial in this phase, as they have the knowledge and the needs of the organization. Natural language models can be considered as stakeholder-oriented models, because their expressiveness and semi informal style allow stakeholders to be involved in their definition. However, the construction of this kind of models consumes time and effort, both crucial and value resources of an organization, discouraging, in consequence, their adoption in a real project. MDD (Model Driven Development) [18] paradigm allows to partially reduce this disadvantage, because it adds a new value to these models: they not only serve to document and communicate the knowledge of the organization but also they become the initial models from which the later models of the software development process can be derived. This added value improves the investment return of the requirements models definition, favoring, in consequence, their incorporation [24].

Model-driven requirements engineering is a relatively novel approach that has started once model-driven conceptual modeling has been proved to be successful [7]. It is concerned with the definition of business knowledge through the CIM (Computation Independent Model) that can be the basis for a MDA (Model Driven Architecture) development. This model represents the system context. Generally this model is obtained manually and it is the first model in MDA development

DOI : 10.5121/ijcsea.2015.5101

process. Although it is not possible to define a completely automatic transformation to obtain a CIM, some works that may contribute in this way have been published, as for example [1, 3, 6, 8, 23].

We have been working in the use of natural language requirements oriented models in an object oriented software development, defining a strategy to derive a class diagram from these requirements models [14, 15]. In order to automate the strategy, some rules have been formalized, choosing ATL [2] as target language. In this way, we provide a transformation from natural language oriented models to a CIM represented with an UML business class diagram [5]. We have continued working in this transformation by adjusting the existent rules and by incorporating a traceability mechanism.

In this work we present an improved version of the ATL transformation that aims to define an UML class diagram representing the structural aspects of a CIM starting from natural language requirements models. With this transformation, we enhance a MDA software development process by obtaining an UML class diagram that will be the basis for a PIM (Platform Independent Model). Many proposals have been made to provide transformations from requirements models to class diagram models in MDA context, such as [1, 3, 8, 23]. The main difference between these proposals and ours is the source requirements model used in the transformation.

The paper is structured as follows: Section II describes the natural language requirements models and contains a brief description of Requirements Engineering in the context of MDD. Section III describes an overview of the transformation process that was improved with a traceability mechanism; Section IV presents the ATL transformation process and the associated metamodels. Section V shows the application of the transformation to a case study. Finally, in Sections VI and VII we present some conclusions and outline possible future work.

2. BACKGROUND

In this section we describe the two natural language oriented requirements models used in this work, and we briefly present the state of the art concerning requirements in MDD.

A. Natural Language Requirements Models

The use of natural language for documenting requirements has several key advantages compared to using a formal specification language, but it also has some key disadvantages [21]. Natural language allows stakeholders document any kind of knowledge concerning the requirements of the system; however, this documentation may be ambiguous because natural language is inherently ambiguous. Some techniques may be applied in order to reduce the ambiguity, such as glossaries, syntactic Requirements Patterns and controlled languages. The models we use in this proposal address the first two techniques. In the original strategy, three natural language oriented models were proposed to define different aspects of the system context: a glossary called LEL (Language Extended Lexicon) [13] to define the terms of the organization, a scenario model [13] to define the behavior, and a business rule model to define the policies of the organization [14]. Up to now, we have incorporated the LEL and Scenario models into the ATL transformation.

Language Extended Lexicon (LEL)

The Language Extended Lexicon, called LEL, is a structure that allows the representation of significant terms of the UofD. The purpose of the lexicon is to help the understanding of the UofD vocabulary and its semantics, leaving the comprehension of the problem for a future step. It

unifies the language allowing communication with stakeholders. LEL is composed of a set of symbols. Symbols are, in general, words or phrases that stakeholders repeat or emphasize. Each symbol has the following structure: symbol name that is a word or phrase and a set of synonyms, notion that describes the denotation of the symbol, and behavioral response or impact describing the symbol connotation, i.e., how does the symbol impact on the UofD. In the description of notions and impacts there are two basic rules that must be followed simultaneously: the "closure principle" that encourages the use of LEL symbols when defining other LEL symbols, and the "minimum vocabulary principle" where the use of external symbols to the UofD language is minimized, and the ones used should refer to a very small and well accepted general core. LEL defines a general classification for the symbol: objects (passive entities), subjects (active entities), verbal phrases and states. Table 1 shows the heuristics to define each type of symbol.

Table 1. Heuristics to describe LEL symbols

Subject	*Notions:* who the subject is.
	Behavioral responses: register actions executed by the subject.
Object	*Notions:* define the object and identify other objects with which the former has a relationship.
	Behavioral responses: describe the actions that may be applied to this object.
Verb	*Notions:* describe who executes the action, when it happens, and procedures involved in the action.
	Behavioral responses: describe the constraints on the happening of an action, which are the actions triggered in the environment and new situations that appear as consequence.
State	*Notions*: what it means and the actions which triggered the state.
	Behavioral responses: describe situations and actions related to it.

Fig. 1 shows an example taken from the case study described in Section 5. Underlined words represent other LEL symbols, following the “closure principle” previously mentioned.

Meeting (Object)
Notion:
- Gathering of people with a goal.
- It has place, date and time established.
- Appears in a meeting schedule.
- It has a list of topics

Behavioral response:
- The participants present the assigned subjects with their material to exhibit.
- The participants hand out the material to distribute.

Requester/Meeting initiator (Subject)
Notion:
- Person who invites potential meeting attendees to a meeting.

Behavioral response:
- Defines the meeting goal, the subjects to be discussed, the potential meeting attendees/attendee, the material to exhibit, and the material to distribute.
- Records the meeting goal and the potential meeting attendees in the agenda.
- Performs the meeting scheduling.
- Organizes the meeting.
- Records the substitute in the agenda.

Figure 1 - Example of LEL Symbols

Scenario Model

A scenario describes UofD situations. Scenarios use natural language as their basic representation. They are naturally connected to LEL. Table 2 describes the components of a scenario.

Table 2 - Scenario Definition

Title: identifies a scenario. In the case of a sub-scenario, the title is the same as the sentence containing the episode. ***Goal:*** describes the purpose of the scenario. ***Context:*** defines geographical and temporal locations and preconditions. ***Resources:*** identify passive entities with which actors work. ***Actors:*** detail entities actively involved in the scenario, generally a person or an organization. ***Set of episodes:*** each episode represents an action performed by actors using resources. An episode may be explained as a scenario; this enables a scenario to be split into sub-scenarios.

Notions of Constraints and Exceptions may be added to some of the components of a scenario. A constraint refers to non-functional requirements and may be applied to context, resources or episodes. An exception, applied to episodes, causes serious disruptions in the scenario, asking for a different set of actions which may be described separately as an exception scenario. Fig. 2 shows one scenario of the proposed case study. Underlined words represent LEL symbols.

Title: **ORGANIZE THE MEETING**
Goal: Guarantee an efficient development of the meeting.
Context: MEETING SCHEDULING must have been done.
Resources: equipment, physical space.
Actors: requester, secretary, potential participants
Episodes:
The requester instructs the secretary about the meeting call.
CALL TO THE MEETING.
NOTIFY ASSISTANCE.
NOTIFY ABSENCE.
[ASK FOR EQUIPMENT.]
[REMIND THE MEETING]
[The secretary assures that the equipment is available for the meeting date. **]**
The secretary assures that the physical space is available for the meeting date.
#

Figure 2 - Example of a Scenario

The scenario construction process starts from the application lexicon, producing a first version of scenarios derived exclusively from the LEL. These scenarios are improved using other sources of information and organized in order to obtain a consistent set of scenarios that represents the application. The complete process is described in [13].

B. Requirements Engineering in Model-Driven Development

There is an agreement about the idea that software requirements definitions have a great impact on final product quality if they are properly managed, well-documented and easily understandable [4]. However, requirements engineering is one of the software engineering disciplines in which model-based approaches are still not well-known. Textual requirements are generally regarded as text fragments that are interpreted and manually managed by stakeholders and developers [16]. A variety of methods and model-driven techniques have been published in literature. However, only a few of them explicitly include the requirements discipline in the Model-Driven Development (MDD) process. The systematic review by [16] analyses several proposals of requirements engineering techniques that have been employed in MDD approaches and their actual level of automation. There are several proposals that integrate different models of RE into MDD, most of them suggest goal oriented models [1,6] or natural language oriented model [12] to define the requirements models that will be the source model for the transformation process that derives, in most cases, UML models. But, the systematic review found a lower level of automation of the

proposed transformations (45% automated transformations against 37% manual, and 17% interactive ones). Traceability is tackled by several proposals but it is still an open problem with several challenges to be addressed [25]. This research is summarized in [7], with the conclusion that model driven requirements engineering is an active research topic with several open research challenges: a) There is a need for technological support for model transformations (automatic or interactive transformations), b) There is a need for better technological support for requirements traceability (guidelines for forward and backward post requirements traceability), and c) There is a need for empirical validations of model driven requirements engineering proposals. Requirements integration with MDD continues growing; for a complete list of approaches refer to [16].

3. AN OVERVIEW OF THE DERIVATION PROCESS

We have been working in the integration of natural language requirements models and object models, proposing a strategy to define a conceptual object oriented model from LEL, scenarios, and business rules. Our proposal begins with the definition of the classes, its methods and attributes starting from the LEL symbols. Then, the Scenario Model is used in order to complete the methods and collaborations between classes. Finally, the functional business rule model is considered to refine the classes and/or to define new classes modeling a set of related business rules.

Relationships between classes are modeled from LEL information. The result of these steps is a conceptual object model that serves as a basis for object-oriented development, reducing the gap between analysis and design phases. The full work may be found in [14]. In [15] we presented five rules for defining an UML class diagram in an automatic way. These rules are a simplification of the derivation process with the objective of defining an automatic transformation aligned with MDA. Derivation rules take LEL and Scenario models as input and define the classes, attributes, methods and relationships necessary to describe a class diagram that defines the structural aspects of a CIM. In order to be aligned with MDA, we defined the ATL transformation for some rules of the strategy [5]. In this work, we improve this ATL transformation by providing a simple trace mechanism that creates a trace relationship between the elements of the source and target metamodels.

A. The Transformation Rules

Rule 1: *Subject to Class Transformation*
Description: LEL subjects correspond to actors in Scenario Model. They represent individuals or part of the organization. Within the scope of conceptual modeling, these entities are those that carry out the main actions of the organization; for this reason, their representation as classes is automatic. Attributes are taken from the LEL notion.

Transformation

- Each LEL subject is transformed into an UML class.
- For each entry in the notion of this type of LEL symbol that does not reference another LEL symbol, the rule identifies each noun and defines it as an attribute.

Rule 2: *Object to Class Transformation*
Description: LEL objects represent significant entities of the UofD, modeling a necessary resource so that subject symbols can perform their behavior. Attributes are taken from the LEL notion and basic methods are defined from them.

Transformation

- Each LEL object is transformed into an UML class.
- For each entry in the notion of this type of LEL symbol that does not reference another LEL symbol, the rule identifies each noun and defines it as an attribute.
- Methods to access and modify each attribute are defined by adding to the name of the attributes the prefixes GET and SET respectively.

Rule 3: *Subject Behavioral Response to Method Transformation*
Description: The behavioral response of subject LEL symbols defines the main actions performed by them in the UofD. As each of these terms was modeled as a class in Rule 1, then, each impact is defined as a method of the corresponding class.

Transformation

- Each entry in the behavioral response of a LEL subject that was modeled as a class by Rule 1, is modeled as a method of that class.

Rule 4: *Subject Information to Method Parameter Transformation*
Description: Scenarios are defined from the entries of behavioral response of LEL subjects; therefore they describe all the necessary data to complete the behavior.

Transformation

- Each scenario comes from an entry in the behavioral response of a LEL subject that was modeled as an UML class.
- The rule models actors and resources of each scenario as parameters of the method obtained by Rule 3 from the entry in the behavioral response that originated the scenario. The actor referring to the subject LEL symbol in consideration is excluded**.**

Rule 5: *LEL Relationships to Class Associations Transformation*
Description: The notion of the LEL symbol describes relationships with other LEL symbols by referencing them in its description. This information is used to define association between classes.

Transformation

- The entries in the notion of each symbol in the LEL modeled as a class are analyzed in order to find other symbols in the LEL modeled as classes.
- For each detected LEL symbol, this rule defines a relationship between the corresponding classes, analyzing the verb involved to determine the type of relationship. In the ATL transformations presented in this paper, inheritance and association relationships are detected.

B. Traceability

Traceability plays a crucial role in requirements engineering [25]. The transformation process we have proposed allows a simple trace mechanism, based on [10] by creating a trace relationship between the source and the target elements of the corresponding metamodel according to each transformation rule. For example, each class in the class diagram is related with one subject LEL symbol or one object LEL symbol because the class was originated from one of them. Each of these relationships has its own semantics, and there may be more than one relationship between those components, depending on the rules. Table 3 shows each trace relationship between elements of the source and target metamodels: the first column represents the source elements and

the top row the target elements. Cells with data indicate a trace relationship. Trace shows the relationships that give rise to new elements in the target metamodel from elements in the source metamodel, i.e. forward relationships, but from them backward traceability ones may be also obtained.

For each trace relationship the following item are described:

- Cardinality of source: how many elements were used to create the new element (Table 3, in the left side of the parenthesis)
- Cardinality of target: how many elements are created in that relationship (Table 3, in the right side of the parenthesis)
- Name of the rule that originated the trace relationship

Table 3 - Trace Relationship between LEL and Scenario Metamodels and the UML Class Diagram generated by the application of the Rules

Target Source	Class	attributes	methods	Method parameter	association
LEL subject	(1/ 0..1) Rule1: Subject to Class				
Noun of LEL subject symbol		(0..1/1) Rule1: Subject to Class			
LEL object	(1/0..1) Rule2: Object to Class				
Noun of LEL object symbol		(0..1/1) Rule2: Object to Class	(0..1/1) Rule2: Object to Class		
Entry in behavioral response of LEL subject			(1/1) Rule3: Subject Behavioral Response to Method		
Actor of scenario				(0..N/1) Rule4: Subject Information to Method Parameter	
resource of scenario				(0..N/1) Rule4: Subject Information to Method Parameter	
Entry in the notion LEL symbol					(0..1/1) Rule5: LEL Relationships to Class Associations

4. FORMALIZING THE TRANSFORMATION PROCESS WITH ATL

This section describes the ATL transformation that obtains a class diagram from LEL and Scenario Model. This ATL definition represents the five rules described before and it is a single

module involving several ATL Rules (both matched and lazy rules) along with a set of helpers. In order to define the transformation and execute it, we must define the source and target models as an Ecore Metamodel. Traceability is also implemented as a separate model, as we describe later in this section.

A. Source and Target Models

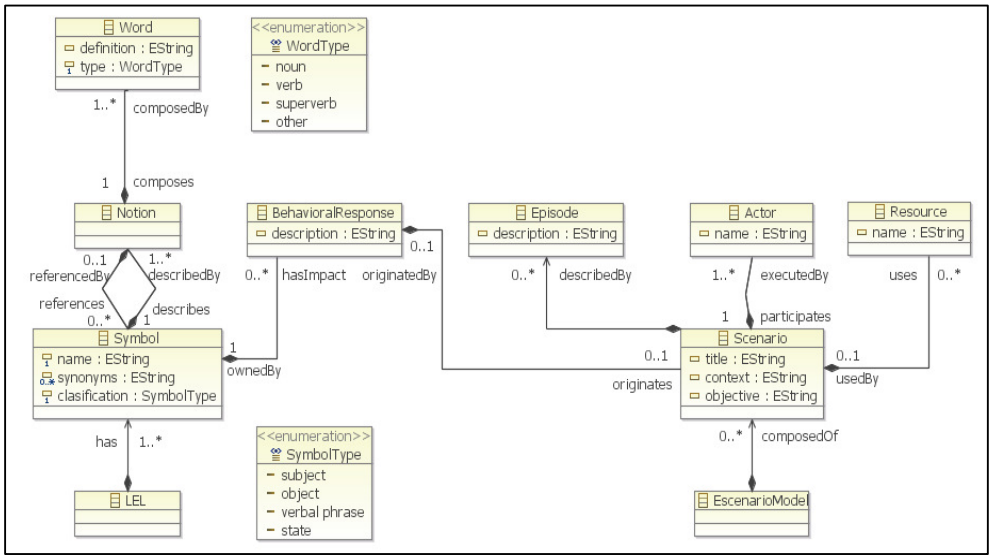

Figure 3 - LEL and Scenarios Metamodel

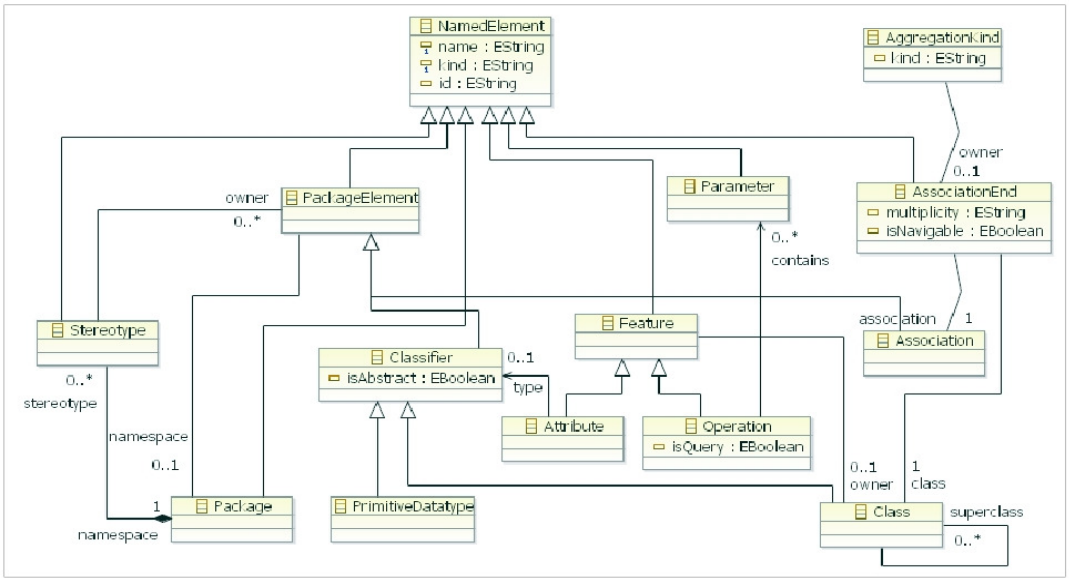

Figure 4 - UML Metamodel

Fig. 3 shows a diagram with the Metamodel of LEL and scenarios defined for the ATL transformation. We have defined an Ecore Metamodel for the UML class diagram by using the simplified version extracted from [22]. Fig. 4 shows this metamodel.

B. The ATL Transformation

In this section, we describe the ATL transformation explaining each of the defined rules and helpers. The transformation contains two matched primary rules that guide the entire process of transformation, the rule Subject2Classand the rule Object2Class. These rules allow the matching of all subject and object LEL symbols and define for each of them an UML class. For each class, the rules identify its name, and perform the corresponding associations with its attributes, its methods and its relationships. To define these attributes, methods and relationships, we define several matched and lazy rules as well as helpers.

RULE Subject2Class: This rule matches subject LEL symbols and associates each of them with an UML class, setting the class name as the name of the symbol. Then, it sets the new class associations that will allow to relate to methods, attributes and relationships that will be defined by the rest of the rules and helpers: a) The helper **getnouns** extracts the nouns from notions of the corresponding LEL symbol. b) The lazy rule **noun2Attribute** defines attributes from nouns that belong to the notion of the LEL symbol which originated the UML class. c) The helper **getSymbolSuper** returns, if any, the symbol that establishes a hierarchy relationship. The UML class corresponding to that symbol will be associated as a superclass to the subclass in **Subject2Class** rule. d) **BehavioralResponse2Method** rule transforms the behavioral response of a Subject to methods. This rule is complemented by the helpers **scenarioresource** and scenarioactor which return the actors and resources of the scenario created from the entry in the behavioral response, and by the lazy rules **Resource2Parameter** and **Actor2Parameter** that transform the actors and resources into the parameters of the defined method. The lazy rule **assocend** defines the associations from the entries in the notion of a symbol that has references to other LEL symbols (we define a helper **getNotionswithRefnonHierarchy**, which returns the entry in the notion of a symbol that refers to other symbols disregarding those that correspond to a hierarchy relationship). This rule is complemented by the helper **getSymbol** that gets the symbol that is part of the association and the helper **firstVerb** that returns the name of the verb which becomes the name of the association. The lazy rule **unnamed_assocend** is used when there is no verb included in the mentioned notion. In this case, the name of the association is set with the concatenation of the names of the involved classes. Fig. 5 presents the ATL code for **Subject2Class** rule.

```
module LELandSce2UML;
create OUT: uml from IN: lelandscenarios;

……

rule Subject2Class {
from
       S1: lelandscenarios!Symbol( S1.clasification=#subject)
to
       C1: uml!Class (
              name <- S1.name,
              features <- S1.getnouns->collect (noun |
              thisModule.Noun2Attribute(noun)),
              features <- S1.hasImpact,
              superclass<- S1.getSymbolSuper,
              associationEnd <- S1.getNotionswithRefnonHierarchy->collect (n |
              S1.getNotionswithVerb -> collect (nv | thisModule.assocend(nv))),
              associationEnd <- S1.getNotionswithRefnonHierarchy->collect (n |
              S1.getNotionswithoutVerb -> collect (ns |
              thisModule.unnamed_assocend(ns)))
       )
}
```

Figure 5 - ATL Code for Subject2Class Rule

RULE Object2Class: This rule matches object LEL symbols and associates each of them with an UML class, setting the class name as the symbol name. Then, it sets the new class associations that will allow relating methods, attributes and relationships that will be defined by the rest of the rules. In addition to the rule **assocend** and its associated helpers, the lazy rule **Noun2Attribute** and helpers **getnouns**} and **getSymbolSuper**, the following rules and helpers are defined in order to complete the definition of a class derived from an object LEL symbol: a) The lazy rule **GetMethod** defines an access method for each attribute of the new class. b) The lazy rule **setMethod** defines a set method for each attribute of the new class. Fig. 6 shows the ATL code for **Object2Class** rule.

```
rule Object2Class {
from
      S1: lelandscenarios!Symbol( S1.clasification=#object)
to
      C1: uml!Class  (
            name <- S1.name,
            features <- S1.getnouns->collect (noun |
            thisModule.Noun2Attribute(noun)),
            superclass<- S1.getSymbolSuper,
            features <- S1.getnouns->collect (noun | thisModule.getMethod(noun)),
            features <- S1.getnouns->collect (noun | thisModule.setMethod(noun)),
            associationEnd <- S1.getNotionswithRefnonHierarchy->collect (n |
            S1.getNotionswithVerb -> collect (nv | thisModule.assocend(nv))),
            associationEnd <- S1.getNotionswithRefnonHierarchy->collect (n |
            S1.getNotionswithoutVerb -> collect (ns |
            thisModule.unnamed_assocend(ns)))
      )
}
```

Figure 6 - ATL Code for Object2Class Rule

C. Implementing the Traceability Mechanism

To implement traceability in our transformation process, we follow Jouault proposal [10]. In this work a trace mechanism is defined by considering traceability information as a separate model, and the code to generate trace relationship is added directly to the transformation rules. Following this idea, we have defined our trace metamodel, as shown in Fig. 7 and added the corresponding code to the rules in order to define the trace relationship presented in Table 3.

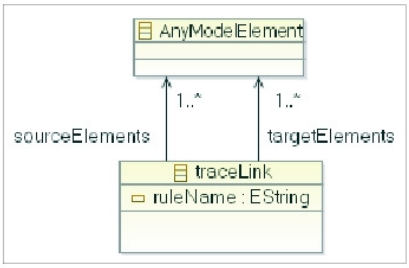

Figure 7- Trace Metamodel

All the matched and lazy rules that create elements of the class diagram are modified in order to define the trace information. We present the rule **Subject2Class**.

```
module LELandSce2UML;
create OUT: uml, trace: Trace from IN: lelandscenarios;

……

rule Subject2Class {
from
        S1: lelandscenarios!Symbol( S1.clasification=#subject)
to
        C1: uml!Class (
                name <- S1.name,
                features <- S1.getnouns->collect (noun |
                thisModule.Noun2Attribute(noun)),
                features <- S1.hasImpact,
                superclass<- S1.getSymbolSuper,
                associationEnd <- S1.getNotionswithRefnonHierarchy->collect (n |
                S1.getNotionswithVerb -> collect (nv | thisModule.assocend(nv))),
                associationEnd <- S1.getNotionswithRefnonHierarchy->collect (n |
                S1.getNotionswithoutVerb -> collect (ns |
                thisModule.unnamed_assocend(ns)))
        ),
        TL1: Trace!traceLink (
                ruleName <- 'SubjectToClass',
                targetElements <- Sequence {C1}
        )
do      {
                TL1.refSetValue('sourceElements', Sequence{S1});
        }
}
```

Figure 8 - ATL Code for Subject2Class Rule including traceability generation

Table 4 shows the implementation of traceability for each transformation shown in Table 3. For example, during the transformation of a LEL object to an UML class, trace information is generated for **getMethod**, **setMethod**, **Noun2Attribute**, **assocend**, and **unnamed_assocend** lazy rules, besides **Object2Class** rule.

Table 4 - Trace Relationship between LEL and Scenario Metamodels and the UML Class Diagram generated by the application of the Rules

Target / Source	**Class**	**Attribute**	**Method**	**Method Parameter**	**Association**
LEL subject	(1/ 1) **rule** Subject2Class				
Noun of LEL subject symbol		(0..1/1) **rule** Subject2Class **lazy rule** Noun2Attribute			
LEL object	(1/1) **rule** Object2Class				
Noun of LEL object symbol		(0..1/1) **rule** Object2Class **lazy rule** Noun2Attribute	(0..1/1) **rule** Object2Class **lazy rule** getMethod **lazy rule** setMethod		
Entry in behaviou			(1/1) **rule** Behavioral		

ral response of LEL subject			Response2Method		
Actor of scenario				(0..N/1) **rule** BehavioralResponse2Method **lazy rule** Actor2Parameter	
Resource of scenario				(0..N/1) **rule** BehavioralResponse2Method **lazy rule** Resource2Parameter	
Entry in notion of LEL symbol					(0..1/1) **rule** Object2Class **lazy rule** assocend **rule** Object2Class **lazy rule** unnamed_assocend **rule** Subject2Class **lazy rule** assocend **rule** Subject2Class **lazy rule** unnamed_assocend

5. APPLICATION OF THE TRANSFORMATION TO A CASE STUDY

We applied the ATL transformation described in this paper to the case study "Meeting Scheduler", as it is a well known one. In particular, we used the version proposed in [9]. Fig. 1 and Fig. 2 were taken from this work.

Figure 9 - Fragment of XMI for Requester LEL Symbol

```
0.      <has name="Requester" clasification="subject">
    0.  <describedBy references="//@has.1 //@has.2">
          0.      <composedBy definition="Person" type="noun"/>
          1.      <composedBy definition="who"/>
          2.      <composedBy definition="invites" type="verb"/>
          3.      <composedBy definition="potential meeting attendees" type="noun"/>
          4.      <composedBy definition="to"/>
          5.      <composedBy definition="a"/>
          6.      <composedBy definition="meeting"/>
                   </describedBy>
                   <synonyms>Meeting initiator</synonyms>
    0.  <hasImpact description="Defines the meeting goal, the subjects to be discussed, the potential meeting
          attendees, the material to exhibit, and the material to distribute"/>
    1.  <hasImpact description="Records the meeting goal and the potential meeting attendees in the agenda"/>
    2.  <hasImpact description="Performs the meeting scheduling"/>
    3.  <hasImpact description="Organizes the meeting">
          0.      <originates title="Organize the Meeting" context="Precondition: MEETING SCHEDULING
                  must have been done" objective="Guarantee an efficient development of the meeting">
            0.      <executedBy name="Requester"/>
            1.      <executedBy name="secretary"/>
            0.      <uses name="equipment"/>
            1.      <uses name="physical space"/>
            0.      <describedBy description="The requester instructs secretary about the meeting call"/>
            1.      <describedBy description="CALL TO THE MEETING"/>
            2.      <describedBy description="[ ASK FOR EQUIPMENT ]"/>
            3.      <describedBy description="[ REMIND THE MEETING ]"/>
            …
                                  </originates>
    4.  </hasImpact>
    5.  <hasImpact description="Records the substitute in the agenda"/>
</has>
```

For this example, we consider the scenario ORGANIZE THE MEETING (Fig. 2) derived, following the derivation Scenario strategy, from one of the entries in the behavioral response of the subject Requester.

These LEL symbols and scenarios were defined in an XMI format to be used as source in the derivation of the UML class diagram. For example, the XMI definition for the symbol Requester is in Fig. 9. The same definition, using the Sample Reflective Ecore Model, is presented in Fig. 10.

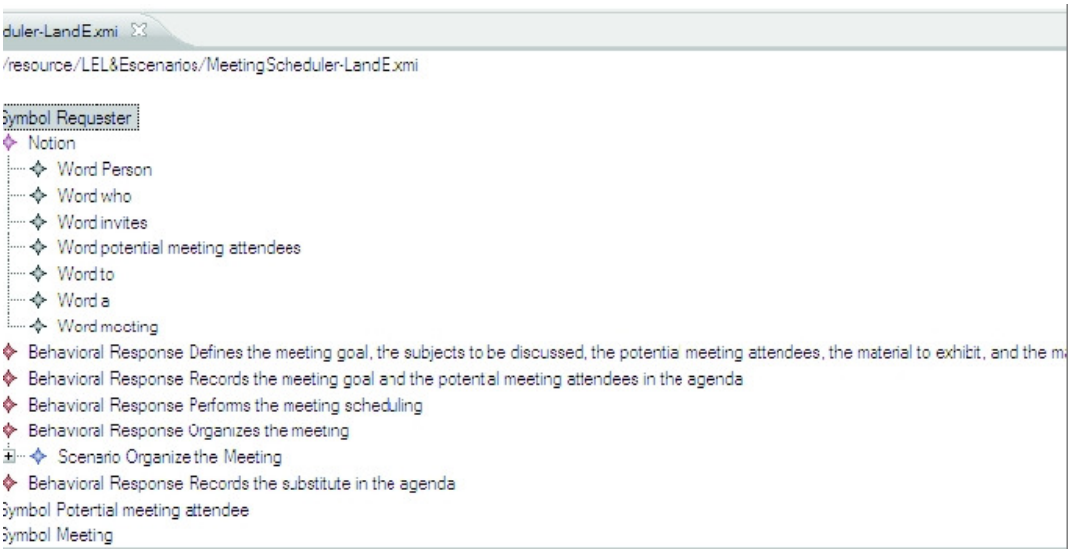

Figure 10 - Sample Reflective Ecore Model for Requester LEL Symbol

Fig. 11 shows the XMI description for one of the UML classes obtained after applying the ATL transformation to LEL and Scenarios for Meeting Scheduler. For the sake of brevity, we have only included the output for the Requester symbol and the scenario Organizes the Meeting.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:uml="http://uml/1.0">
0  <uml:Operation name="Defines the meeting goal, the subjects to be discussed, the potential meetingattendees, the
       material to exhibit, and the material to distribute" owner="/18"/>
1   <uml:Operation name="Records the meeting goal and the potential meeting attendees in the
       agenda"owner="/18"/>
2   <uml:Operation name="Performs the meeting scheduling" owner="/18"/>
3   <uml:Operation name="Organizes the meeting" owner="/18" contains="/21 /22 /23 /24"/>
4   <uml:Operation name="Records the substitute in the agenda" owner="/18"/>
…
18 <uml:Class name="Requester" features="/0 /1 /2 /3 /4" associationEnd="/52/@associationEnd.0"/>
…
21 <uml:Parameter name="equipment"/>
22 <uml:Parameter name="physical space"/>
…
52 <uml:Association name="invites_asoc">
            <associationEnd name="ae_Requester" isNavigable="true" class="/18"/>
            <associationEnd name="ae_Potential meeting attendee" class="/19"/>
      </uml:Association>
…
</xmi:XMI>
```

Figure 11 - Fragment of XMI for Meeting Scheduler UML Model

Fig. 12 presents an extract of the trace information also produced after the ATL transformation for the symbol Requester and the scenario Organizes the Meeting. Each trace link includes a reference to an element in the source model (LEL and Scenarios), and another one to an element in the target (UML Model). For example, tracelink with ruleName **SubjectToClass** has a reference to the LEL symbol 0, shown in Fig. 9, as sourceElement, and another reference to UML class 18, shown in Fig. 11, as the targetElement.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:trace="http://trace/1.0">
<trace:traceLink ruleName="SubjectToClass">
  <sourceElements href="MeetingScheduler-LandE.xmi#//@has.0"/>
  <targetElements href="MeetingScheduler-UML.xmi#/18"/>
</trace:traceLink>
…
<trace:traceLink ruleName="BehavioralResponseToMethod">
   <sourceElements href="MeetingScheduler-LandE.xmi#//@has.0/@hasImpact.0"/>
   <targetElements href="MeetingScheduler-UML.xmi#/0"/>
</trace:traceLink>
…
<trace:traceLink ruleName="ResourceToParameter">
      <sourceElements href="MeetingScheduler-LandE.xmi#//@has.0/@hasImpact.3/
           @originates/@uses.0"/>
  <targetElements href="MeetingScheduler-UML.xmi#/21"/>
</trace:traceLink>
…
```

Figure 12 - Fragment of XMI for Meeting Scheduler Trace Model

6. DISCUSSING THE TRANSFORMATION PROCESS

The execution of the ATL transformation rules allows an automatic definition of a tentative object-oriented CIM. Though a manual derivation produces a better and more accurate model definition [14], ATL transformation is a starting point to deal with the great amount of requirements information by providing a systematic and consistent way of defining CIM´s in MDA framework. The CIM should be later refined by a software engineer, who will correct and complete it.

Considering our experience with manual derivation strategies and the automatic transformation we propose in this paper, we want to discuss the following issues:

- Our proposal is mainly based in the metamodel of LEL. The transformation rules were defined considering the way in which the concepts of the Universe of Discourse are described, explicitly defining structural and behavioral aspects of them. For example, definition of classes is based on the classification of LEL symbols, automatically modeling one class per each subject or object LEL symbol. The strategy to find methods and parameters is also based on the structure of the models. However, to identify attributes we have to analyze the text of notions. In this first approach, we follow a basic linguistic strategy to find nouns in notion. One of the main problems is that this mechanism misses groups of nouns. As the helper **getnouns** only detects separate nouns, every noun is a potential attribute, thus generating more and sometimes inappropriate attributes. However, groups of nouns detection may be included following linguistic approaches [11].
- We think the free style to express the content of notions and behavioral responses of LEL symbols makes difficult the automatic processing of the information they describe. Manual

heuristics could use human intelligence to take the final decision. In some cases, it would be possible to define a standard form of writing without restricting the power of expression of natural language [21].

- Though LEL and scenarios have a precise structure, the use of natural language allows the same semantics to be usually expressed with many different natural language sentences. For example, in some cases the same concept may be described with a noun or a verbal phrase since each essential concept has a root expression as a noun, a verb or even as an adjective [11]. The manual strategies already mentioned use human judgment to decide if a verbal phrase should be modeled as a class or as a method. An automatic transformation takes always the same decision loosing, in some cases, the real meaning of the essential concept. In our proposal LEL verbal phrases remain as methods of classes modeling subject LEL symbols because they appear as entries of the behavioral responses of the Subject LEL symbol. We take this decision to avoid the definition of classes with only one method, as advised in [17]. Later, this may be modified by the software engineer.

7. CONCLUSIONS

Natural language oriented models are widely used in requirements modeling due to their well-known advantages [21]. They allow representing the system context as a conceptual model, a crucial issue for the success of a software system development. This kind of requirements models have to be reinterpreted by software engineers into a more formal design on the way to a complete implementation, and in the context of MDA this reinterpretation may be done as model transformation. Therefore, an automatic transformation to map their knowledge into conceptual object models would be really useful. Several approaches have been proposed in order to incorporate Requirements Engineering into MDA, for example [1, 2, 3, 12, 23]. In this paper we propose a natural language requirements engineering approach for MDA, and we present an ATL transformation to obtain a CIM starting from natural language oriented requirements, more concretely LEL and scenario models. This CIM serves as the basis for the development of software systems, aligning our transformation process with the MDA framework. The main difference with existent approaches is in the natural language requirements models used as source models in our proposal.

We must refine some ATL rules in order to obtain a better definition of the relationships between UML classes, implementing the manual strategy proposed in [14]. Also, in the manual strategy, we have another model, the Business Rule Model to define the policies that govern the structure and behavior of the organization. We plan to incorporate this model into the ATL transformation, extending the requirements metamodel and defining the corresponding ATL transformation rules.

Traceability plays a crucial role in requirements models and in MDD [25]. The transformation process we have proposed and implemented in the ATL rules allows the trace between the source and the target. Trace information is easily created but, until the moment it is not managed. We must incorporate trace supporting in order to recorded traces became useful for the entire development process.

We accept that LEL and scenarios models are non-standard notations, thus reducing a broader use of them. For this reason, we plan to adopt the Business Process Modeling Notation (BPMN) [19] to support scenario models and SVBR [20] to model the Lexicon and to adapt the transformation process. But a careful investigation needs to be carried out to adopt the notation while preserving the philosophy that scenario and LEL models is based on.

OMG is actively working in projects concerning business modeling and MDA. These results will help us to enhance our proposal.

REFERENCES

[1] Alencar F, O. Pastor, B. Marín, G. Giachetti, J. Castro, and J. Pimentel (2009). "From i* Requirements Models to Conceptual Models of a Model Driven Development Process". Lecture Notes in Business Information Processing, Vol.39, Part 4. pp.99-114.

[2] ATL A Model Transformation Technology (2014). Last access September 2014 at http://www.eclipse.org/atl/.

[3] Bousetta Brahim, EL Beggar Omar, GADI Taoufiq (2013). "A methodology for CIM modelling and its transformation to PIM" Journal of Information Engineering and Applications www.iiste.org. ISSN 2224-5782 (print) ISSN 2225-0506 (online) Vol.3, No.2.

[4] Davis A.M., Dieste O., Hickey A., Juristo N., and Moreno, A.M (2009). "Effectiveness of Requirements Elicitation Techniques: Empirical Results Derived from a Systematic Review". Proceedings of the 14th International Requirements Engineering Conference. pp.176-185.

[5] Debnath N, Leonardi M.C., Ridao M, Mauco M.V, Felice F, Montejano G, and Riesco D (2009). "An ATL Transformation from Natural Language Requirements Models to Business Models of a MDA Project". Proceedings of 11th International Conference on ITS Telecommunications, Russia. pp.633-639.

[6] Decreus K, Snoeck M, Poels G (2009). "Practical Challenges for Methods Transforming i* Goal Models into Business Process Models". 17th IEEE International Requirements Engineering Conference. pp. 15-23.

[7] España S. (2011)."Methodological Integration Of Communication Analysis into a Model Driven Software Development Framework". Doctoral Thesis, Departamento de Sistemas Informáticos y Computación, Universitat Politécnica de Valencia. http://www.upv.es/entidades/DSIC/menu_urlc.html?/pls/oalu/sic_ted.infoent_menu?P_IDIOMA=C&P_VISTA=MS.

[8] González A, España S , Ruiz M, and Pastor O (2011). "Systematic Derivation of Class Diagrams from Communication in Business-Process and Information Systems Modeling". Proceedings of 12th International Conference Enterprise, Business-Process and Information Systems Modeling, BPMDS 2011, and 16th International Conference EMMSAD 2011, CAiSE 2011, London, UK. pp.246-260.

[9] Hadad G. "Uso de Escenarios en la Derivación de Software" (Use of Scenarios in Software Derivation) (2008). Doctoral Thesis. Universidad Nacional de La Plata, Argentina.

[10] Jouault F (2005). "Loosely Coupled Traceability for ATL". Proceedings of the European Conference on Model Driven Architecture, Workshop on Traceability, ECMDA, Nuremberg, Germany. pp.29-37.

[11] Juristo N., Juzgado, Moreno A.M., and López M (2000). "How to Use Linguistic Instruments for Object-Oriented Analysis." IEEE Software Vol.17, No.3, pp.80-89.

[12] Kleiner, M. (2009): "ATL Use Case - Production of UML Class Diagrams from Syntactical Models of English Text or SBVR Models". Last access September 2014 at on http://www.eclipse.org/atl/usecases/Syntax2SBVR2UML/.

[13] Leite, J.C.S., Doorn J., Kaplan G, Hadad G, and Ridao. M (2004). "Defining System Context using Scenarios". Perspectives on Software Requirements, Kluwer Academic Press. pp.169-199.

[14] Leonardi M.C (2003). "Enhancing RUP Business Model with Client-Oriented Requirements Models". Chapter 6, UML and the Unified Process, IRM Press, USA. pp.80-115.

[15] Leonardi M.C, Mauco M.V (2009). "Integrating Natural Language Requirements Models with MDA". Encyclopedia of Information Science and Technology, Second Edition, IGI Global, USA. pp.2091-2102.

[16] Loniewski, E. Insfran, and S. Abrahão (2010). "A Systematic Review of the Use of Requirements Engineering Techniques in Model-Driven Development,Model Driven Engineering Languages and Systems". Lecture Notes in Computer Science, Vol.6395. pp.213-227.

[17] Meyer B (1997). Object-oriented Software Construction, Prentice Hall.

[18] Model Driven Architecture (2014). Last access September 2014 at http://www.omg.org/mda.

[19] OMG Business Process Modeling Notation (BPMN) version 2.0.2 (2014). Last access July 2014 at http://www.omg.org/spec/BPMN/2.0.2/.

[20] OMG Semantics of Business Vocabulary and Rules (SBVR) (2014). Last access July 2014 at http://www.omg.org/spec/SBVR/1.2.

[21] Pohl K. (2010). "Requirements Engineering: Fundamentals, Principles and Techniques". Springer-Verlag,Berlin.

[22] Pons C, Giandini R, and Pérez G (2010). "Desarrollo de Software dirigido por Modelos - Conceptos Teóricos y su Aplicación Práctica" (Model driven Software Development: Concepts and Applications). EDULP & McGraw-Hill Education.

[23] Raj A, T.V. Prabhakar, and S. Hendryx (2008). "Transformation of SBVR Business Design to UML Models". Proceedings of the 1st Conference on India Software Engineering Conference, ISEC '08, ACM, New York, USA. pp.29-38.

[24] Ruiz M, S. España, A. González, O. Pastor (2010). "Análisis de Comunicaciones como un Enfoque de Requisitos para el Desarrollo Dirigido por Modelos." Actas de los Talleres de las Jornadas de Ingeniería de Software y Bases de Datos, España. Vol2, pp.70--77.

[25] Winkler S, von Pilgrim, J (2010). A Survey of Traceability in Requirements Engineering and Model-Driven Development. Software and Systems Modeling, Vol.9, No.4. pp.529-565.