Automatic Generation of Sequence Diagrams and Updating Domain Model from Use Cases

Fabio Cardoso de Souza, Fernando Antonio de Castro Giorno
Master's Program in Software Engineering
Institute for Technological Research (IPT)
São Paulo, Brazil
e-mail: souzafc@yahoo.com, giorno@pucsp.br

***Abstract*—Software modeling allows for problem decomposition in a way that facilitates analysis and communication of the solution to developers and other interested parties. Models are widely used in engineering in general, but in Software Engineering modeling has often been left out due to the pressures to improve deadlines. A method and a tool that reduce the duration of this phase could help furthering the modeling phase. Use Cases are commonly utilized for functional specifications in Object-Oriented paradigm and the use of markups in Use Cases allow an automatic partial generation of Analysis Models, reducing the time of the modeling phase in this paradigm. This paper proposes a combination of rules for marking up Use Cases and one procedure for generating partial Sequence Diagrams with analysis classes (one Sequence Diagram for each Use Case) and the updating of the Domain Model with operations. A tool was built to prove the concept and two experiments were carried out.**

Keywords-Analysis Model; Use Case; Sequence Diagram; Model Driven Architecture.

I. INTRODUCTION

Software modeling permits the analyst to break the problem to be solved into parts which can be better analyzed. It also allows the formal communication of a functional and technical solution based on the demanded requirements. Model is a formal specification of the structure or function of a system [1]. A graphic representation can be used to provide a visual body for the model.

Despite being widely used in many areas of engineering, modeling has been left out in Software Engineering. According Rosenberg and Stephens [2], in practice, there never seems to be enough time to do modeling, analysis and design and there is always pressure from management to jump to code prematurely because progress on software projects tends to get measured by how much code exists, leading to problems in the quality of software.

Use Cases are commonly used for functional specification in Object-Oriented developments. According to Sommerville [3], Use Cases are an effective technique for eliciting requirements and they are increasingly used since the Unified Modeling Language (UML) became a standard for Object-Oriented modeling. Yet according to Rosenberg and Stephens [2], the Use Cases are created over a Domain Model since this offers the use of a common vocabulary. The utilization of markups in Use Cases can allow for the automatic partial generation of the Analysis Model, as demonstrated in the Mason and Supsrisupachai [20] work, where marked up Use Cases are automatic transformed into Sequence Diagrams.

The automatic partial generation could reduce the duration of the modeling phase, thus stimulating the adoption of this phase in Object-Oriented development projects, as suggested by a qualitative research [23] carried out with requirements analysts, system analysts and project managers. In this qualitative research, the majority of the interviewees agreed that software modeling improves the quality of the final product and most of them believe that the automatic generation of the partial Analysis Model can help the adoption of the modeling phase in software development projects. Due to space limitation, details of this research are omitted.

This paper presents a set of rules for marking up Use Cases and a transformation procedure that permits deriving Sequence Diagrams with analysis classes from the marked-up Use Cases. It also permits the updating of the Domain Model with operations identified in the Sequence Diagrams, leading to the Class Diagram. Class Diagram and Sequence Diagrams are the main diagrams in an Object-Oriented analysis model. The diagrams generated do not take into consideration details of a possible implementation, which must be done during the design phase. According to Booch et al. [4], the analysis must yield a statement of what the system does, not how. This research also presents a tool which implements the proposed procedure and with which the experiments were realized.

The rest of this paper is organized as follows. Section 2 presents concepts on which this research is based. This section also presents the State of the Art in the topics Model Driven Architecture and transformation of Use Cases into Sequence Diagrams. Section 3 presents a proposal for marking up Use Cases and a transformation procedure. Section 4 presents the tool and two experiments. The fifth and final section presents the conclusion and suggestions for future researches.

 ISBN: 978-1-61208-449-7

II. CONCEPTS AND STATE OF THE ART

This section starts presenting concepts related with Use Cases, Software Modeling and transforming requirements into software models approaches, and ends with state of the art on transformation subject.

A. Requirements Specification with Use Cases

Requirements Engineering provides appropriate mechanisms for [5]: understanding what the client wants; analysis of her/his needs; evaluation of feasibility; negotiation of a reasonable solution; specification of requirements in a unambiguous manner; validation of the specification and management of the requirements to be implemented.

Use Cases serves as functional specifications of requirements in Object-Oriented paradigm and the Analysis Model is created based on them. Use Cases provide the external behavior expected by the system with respect to the vocabulary in a Domain Model. Rosenberg [2] states that Use Cases describes a way by which the users interacts with the system and how the system responds. Pressman [5] notes that Use Cases does not tell how a system should realize the functionality. This emphasizes the importance of modeling.

According to Larman [6], Use Cases can be essential or concrete. Essential Use Cases do not consider mechanism details (like User Interfaces), while Concrete Use Cases consider them. In this paper, only Concrete Use Cases are contemplated.

Yet, according to Rosenberg [2], Use Cases should be written in the objects model context, referencing domain classes and boundary classes by their names. This recommendation is the base for this work as the objects constituents of the Analysis Model are the objects referenced in the Use Cases and existing in the Domain Model.

Use Cases makes explicit not only the objects involved in the system boundary but also the actors participating in the functionality and their actions. An actor is any entity that communicates with the system and is external to it, and may be a device, a system or a person. A main actor is that which interacts with the system in order to produce the result while secondary actors only support the system [5].

B. Analysis Model

Modeling is generally done in two levels of abstraction: Analysis Model and Design.

The Analysis Model - or Software Architectural Design - is used to identify, in a high level of abstraction, the components of the software, describing how the software is decomposed and organized into components [7]. In the case of Object-Oriented software, these components are Analysis Classes with their attributes and operations. In this paper, a partial Analysis Model is the expected result of the application of the proposed method.

The Class Diagram is the most important diagram of the Analysis Model and it describes the static vision of the system in terms of classes and relationships between them.

Jacobson [8] distinguishes the following types of classes used to give structure to Object-Oriented software: boundary, control and entity. According to him, boundary classes respond to information and behaviors related to system boundary; entity classes respond to information that are stored in the system and to behaviors surrounding these information; and control classes respond to behaviors which are not naturally incorporated into entities. These definitions are complemented by Bruegge and Dutoit [9], for whom, boundary classes represents interfaces between systems and actors, and control classes are in charge of realizing Use Cases.

The boundary and control classes, as well as their behaviors (their operations) are evident during analysis, in Analysis Model.

In this paper, these three types of objects are adopted in a way through which the Sequence Diagram can represent the software model with these three layers (boundary, control and entities).

Sequence Diagram is the second most important diagram of an Analysis Model and it is used to illustrate how objects interact with one another through messages, demonstrating the internal behavior of one system functionality (one Use Case).

The sequence of messages in a Sequence Diagram can use a pattern of communication between the objects, how, for example, the pattern presented by Heinemann and Denham [10], where messages should follow the flow "boundary ←→ control ←→ entity". This pattern is adopted in this work.

C. MDD and MDA

Model Driven Development (MDD) refers to the approaches based on models as the main products of a development [11]. According to Milicev [12], MDD raises the level of abstraction in a development.

Model Driven Architecture (MDA) is a MDD approach proposed by the Object Management Group (OMG) whose objective is to alleviate the problem of ruptures between design and code due to system migration from one platform to another [11].

MDA advocates four layers of model: Computation Independent Model (CIM), Platform Independent Model (PIM), Platform Specific Model (PSM) and Implementation Specific Model (ISM) as shown in Figure 1.

In the CIM layer there lies the process models and requirements that are independent of computing. In the PIM layer there lies the Analysis Model which is in the computing field, therefore totally independent of platform. In the PSM layer there lies the lower level models, which takes into consideration the platform where the system would be introduced. Finally, the ISM layer is the layer where the code is generated. The development focuses, on the MDA approach, is at a high level of abstraction, that is, in the CIM and PIM layers.

 ISBN: 978-1-61208-449-7

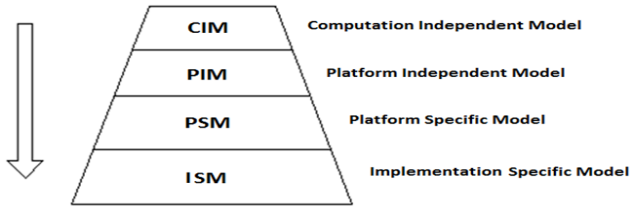

Figure 1. Layers of MDA.

D. Related Work

In [13] a process for generating a model on the CIM layer from the requirements written in natural language was proposed. The requirements should be represented in Language Extended Lexicon (LEL) and in a scenario Model. LEL is a structure that permits representation of significant symbols in the universe of discourse, their synonyms and their behavior. The symbols can be: People, Objects, States, Events, among others. The process consists of a series of transformation rules over texts written in natural language contained in the LEL and in the scenarios.

In [14] a use case modeling approach was proposed in a way that elements of the Use Case are inserted into specific fields of a template, but there are no fields for components of the steps (sender object for example). Under this proposal, the steps should be restricted by a combination of grammar rules and rules for key words utilization. Based on this, the same authors [15] proposed a tool named aToucan (Automated Transformation of Use Case Model into Analysis Model). The tool aToucan reads the restricted steps of Use Cases and realizes the processing of natural language written in steps in order to obtain classes and relationships for the Analysis Model. The result is a generation of an intermediate Unified Modeling Language (UML) meta-model that is then transformed into a final Analysis Model. Only Class Models are mentioned in the obtained results.

In [16][17], it was proposed a set of marking-up rules and a set of syntactic structures in a manner an analyzer can extract the elements in order to generate a Sequence Diagram. The marking-up rules aim to permit the analyst to mark up occurrences of links, conditions and parallelism. The author names the marked-up Use Case with syntactic restrictions by Normalized Use Case.

The analyzer utilizes a dictionary to localize and store the elements in a catalogue applying syntactic rules. The catalogue is then used to obtain, in each message, the object sender, object receiver, operations and arguments that are registered in a file. There is no diagram generation. According to the author, the results needs to be refined by the analyst due to the confusion the analyzer can do while extracting concepts. The Use Case should be written in English natural language.

In [18] a set of transformation rules and a syntactic structure of the steps were also proposed. The steps should be written in this syntactic structure: “Who does What for Who”, being that the first ‘Who’ denotes the actor that starts the communication, the ‘What’ denotes the message to be transmitted, and the second ‘Who’ denotes the receiver of the message. The proposal contemplated a tool for editing Use Case and for generating the Sequence Diagram. The authors consider the method and the tool only as an instrument for learning.

In [19], it was proposed a tool for generation of Sequence Diagrams from Use Cases written in English. The tool uses a pre-existing component (Stanford Parser) to generate parts of speech tagged sentences and type dependencies. It then applies a proposed sentence structure rules and transformation rules to identify elements to generate the Sequence Diagram. The approach works only for the Simple Sentences in English.

In [20], Mason and Supsrisupachai proposed markups to indicate the primitives in a Use Case that derive elements to the respective Sequence Diagram. Only main scenarios of Use Cases are analyzed and each step of a Use Case needs to be marked up with an event type. A data dictionary is utilized as a reference of the Use Case elements. The marking up is made at each step of the Use Case on the elements: object sender, message, object receiver, actions and event timer. A tool was built for editing and marking up Use Cases and for generating the corresponding Sequence Diagram.

E. Consideration on State of the Art

In the MDA field, a lack of an official meta-model defined by the OMG for specification of Use Cases resulted in the presented proposals not fitting exactly into the MDA philosophy, which advocates, among other things, the use of UML and its meta-models as the origin and targets for transformations.

In direct Use Cases transformation into Sequence Diagrams, The Mason and Supsrisupachai [20] work offers a greater precision in the generation of a partial Analysis Model because the analyst previously identifies the elements in the Use Cases, as long as, he understands the problem and can deal with the imprecision of the natural language in an appropriate manner.

III. RULES FOR MARKING UP USE CASES AND THE TRANSFORMATION PROCEDURE

Mason and Supsrisupachai [20] work served as an inspiration for this proposal. As was previously mentioned, the use of markups in Use Cases is an efficient approach for partial automatic generation of Sequence Diagrams with analysis classes.

Some important differences in this work compared to the Mason and Supsrisupachai [20] work are:

This work proposes the updating of Domain Model with the operations identified during the method execution. It uses stereotypes to represent the types of classes according to their layers (boundary, control and entity). They also present a transformation procedure from Use Cases into Analysis Model and, finally, they define markups for actor, interface and guard condition. The set of markups was defined to allow, following the method, the generation of sequence diagrams considering stereotypes, actors (primary and secondary) and messages with condition guards. Even though this automatic generation is not enough for the analyst to start lower level design or code, it can be useful to

 ISBN: 978-1-61208-449-7

the analyst since he does not need to start the modeling from scratch, thus reducing the duration of this phase. The mark-up process is considered to be made, by the analyst, against the Domain Model and trying to use, as much as possible, all the markups. For example, if in a specified step, the interface is not specified, and considering that there is a markup for interfaces, the professional must verify the possibility to explicit an interface in this step.

A. Marking up Rules

Table I below presents a set of markups proposed in this paper.

TABLE I. USE CASE MARKUPS

Markup	***Markup target***	***Markup format***
sdr	Sender object	[sdr Sender]
rcv	Receiver object	[rcv Receiver] or [rcv Receiver: name on Domain Model]
msg	Message	[msg Message] or [msg Message: label]
act	Internal action of the object (recursive message)	[act Message]
a1	Main actor	[a1 Actor]
a2	Secondary actor	[a2 Actor]
ifc	Human–machine or machine-machine interface	[ifc Interface]
grd	Guard condition	[grd condition]

The ‘msg’ markup allows an optional format with the use of a second argument (an optional label) which denotes that the label should be used on the diagram in the place of first argument (the event).

In the same way, “rcv” markup permits an optional format to specify the name of the receiver object when the name used in the step does not reflect the name in the Domain Model.

B. Transformation Process

According to Rosenberg and Stephens [2], Use Cases must be written in the context of the Domain Model, referencing the domain classes and boundary classes by their names. They recommend yet that the steps should be written with the structure: object – verb – object. Sequence Diagrams are behavioral models that illustrate how the objects interact with each other. These interactions are considered, initially (on the partial Sequence Diagram), a representation of the verbs specified on the Use Cases.

As mentioned above, the proposed procedure considers the types of object (boundary, control and entity), the actors and the messages between them with optional condition guard, in order to produce a partial Analysis Model. The following premises are considered in order to identify these elements in the Use Case text:

1. A Use Case step should contain only one message.
2. A step should be in one of the following configurations:
   - 2.1. “Xxx [a1 name] xxx [msg name] xxx [ifc name] xxx.”
   - 2.2. “Xxx [sdr System] xxx [msg name] xxx [ifc name] xxx.”
   - 2.3. “Xxx [sdr System] xxx [act name] xxx.”
   - 2.4. “Xxx [sdr System] xxx [msg name] xxx [a2 name].”
   - 2.5. “Xxx [sdr System] xxx [msg name] xxx [rcv name] xxx.”

Where ‘xxx’ represents free and non-obligatory texts and ‘name’ represents the name of an actor, object or message. A guard condition is optional and may occur in any of the above configurations.

Below is presented the procedure for transforming Use Cases into Sequence Diagrams and for updating the Domain Model with operations.

1. For each Use Case document:
   - 1.1. Is created a Sequence Diagram with the same name as the Use Case.
   - 1.2. Is added, into the diagram, the main actor, the «boundary» classes from ‘ifc’ markups without repetition, a «control» class with the same name as the Use Case, «entity» classes in the same sequence which they occur in the Use Case without repetition, and the secondary actors. «entity» classes are the other objects in Use Case which are neither actors, nor interface nor System.
   - 1.3. For each step in one expected configuration, the specific rules for messages creation must be observed. In any expected configuration, there may be a guard condition.
     - 1.3.1. “Xxx [a1 name] xxx [msg name] xxx [ifc name] xxx.”:
       - 1.3.1.1. One message is created from the main actor to the interface specified in the step.
       - 1.3.1.2. The focus is placed at the interface specified in the step in manner that the next message originates from it.
     - 1.3.2. “Xxx [sdr System] xxx [msg name] xxx [ifc name] xxx.”:
       - 1.3.2.1. If the control object has the focus:
         - 1.3.2.1.1. One message is created from the control object to the interface specified in the step.
         - 1.3.2.1.2. The focus is placed at the interface specified in the step in a way that the next message originates from it, unless there is a guard condition, because in this case, the guard condition may not occur, so the control object continues originating messages.
       - 1.3.2.2. If an interface has the focus:
         - 1.3.2.2.1. One message is created from the interface that has the focus to the interface specified in the

 ISBN: 978-1-61208-449-7

step. This represents a hyperlink from the first interface to the second, and such type of operation does not need to pass through the control object.

1.3.2.2.2. The focus is placed at the interface specified in the step in a manner that the next message originates from it, unless there is a guard condition, because in this case, the guard condition may not occur, so the first interface continues originating messages.

1.3.3. "Xxx [sdr System] xxx [act name] xxx.":

1.3.3.1. If the control object has the focus:

1.3.3.1.1. One recursive message is created in the control object.

1.3.3.1.2. The focus remains at the control object.

1.3.3.2. If an interface has the focus:

1.3.3.2.1. One message is created from the interface that has the focus to the control object.

1.3.3.2.2. One recursive message is created in the control object.

1.3.3.2.3. The focus is placed at the control object.

1.3.4. "Xxx [sdr System] xxx [msg name] xxx [a2 name] xxx.":

1.3.4.1. If the control object has the focus:

1.3.4.1.1. One message is created from the control object to the secondary actor.

1.3.4.1.2. The focus remains on the control object because it is not expected that a secondary actor can originate a message on the next step (secondary actors only supports the system and any activity that it can do is outside of the functionality scope).

1.3.4.2. If an interface has the focus:

1.3.4.2.1. One message is created from the interface that has the focus to the control object.

1.3.4.2.2. Other message is created from the control object to the secondary actor.

1.3.4.2.3. The focus is placed at the control object because it is not expected that a secondary actor can originate a message on the next step (secondary actors only support the system and any activity that it can do is outside of the functionality scope) and the control object originated the last message.

1.3.5. "Xxx [sdr System] xxx [msg name] xxx [rcv name] xxx.":

1.3.5.1. If the control object has the focus:

1.3.5.1.1. One message is created from the control object to the receiver object.

1.3.5.1.2. The focus remains on the control object because it is not expected that a receiver object (an entity by exclusion) can originates a message on the next step (Use Case do not explain the internal behavior of the functionality).

1.3.5.2. If an interface has the focus:

1.3.5.2.1. One message is created from the interface that has the focus to the control object.

1.3.5.2.2. Other message is created from the control object to the receiver object.

1.3.5.2.3. The focus is placed on the control object because it is not expected that a receiver object (an entity by exclusion) can originate a message on the next step (Use Case do not explain the internal behavior of the functionality) and the control object originated the last message.

2. The Domain Model is updated with operations identified in «entity» objects.

C. Limitations

The set of marking-up rules and the transformation procedure has the following limitations:

- Only main scenarios of Use Cases are considered;
- There is not treatment for inclusion and extension relationships at Use Cases;
- Only synchronous messages are considered;
- Message parameters are not considered;
- Loops and parallelism (concurrent processes) are not considered;
- Only Concrete Use Case is considered.

These limitations imply needs for adjustments and complements at the Analysis Model generated by the tool that implements the procedure, in order for the model to be useful for the next phases of the project (design phase and coding).

IV. TOOL AND EXPERIMENTS

Below, we present the tool that implements the proposed procedure and two experiments.

A. Tool

A tool that automates the proposed procedure was developed using Java language and the Netbeans Integrated Development Environment (IDE). The tool is executed as a

Netbeans plug-in and it is presented as a tab on it, where the path to the Use Cases and Domain Model to be processed should be informed. Furthermore in this paper, file formats expected by the tool are presented. Figure 2 shows an overview of the transformation process.

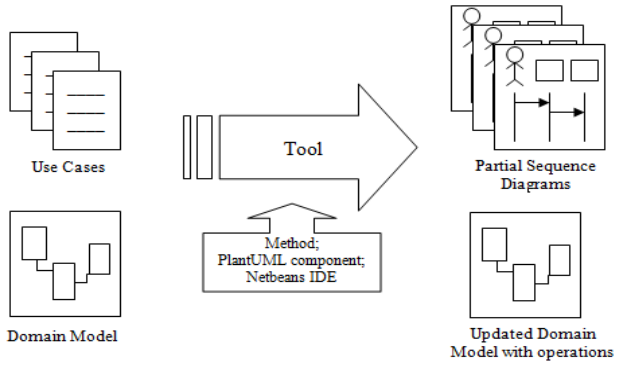

Figure 2. Transformation process overview.

Figure 3 presents an example of a marked-up part Use Case. For each generated diagram by the tool, a new tab is opened in the Netbeans IDE, containing the image of the diagram, as shown in Figures 4 and 5.

Use Case: Rent a Car.

Description: This Use Case describes the steps to make a car reservation on the Vehicle Rental web page.

Main Actor: [a1 Client]

Main Scenario:
1)**The** [a1 **Client**] [msg **request a reservation**: doReservation] **at the** [ifc **VehicleRentalPage**].
2)**The** [sdr **System**] [msg **request identification number**] **at the** [ifc **VehicleRentalPage**].
3)**The** [a1 **Client**] [msg **inform the identification number**: inputIdentificationNumber] **at the** [ifc **VehicleRentalPage**].
...

Figure 3. Example of a marked-up part Use Case.

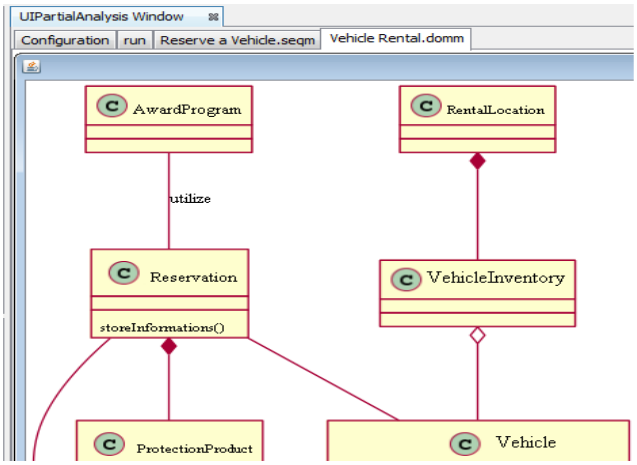

Figure 4. Class Diagram tab (partial view).

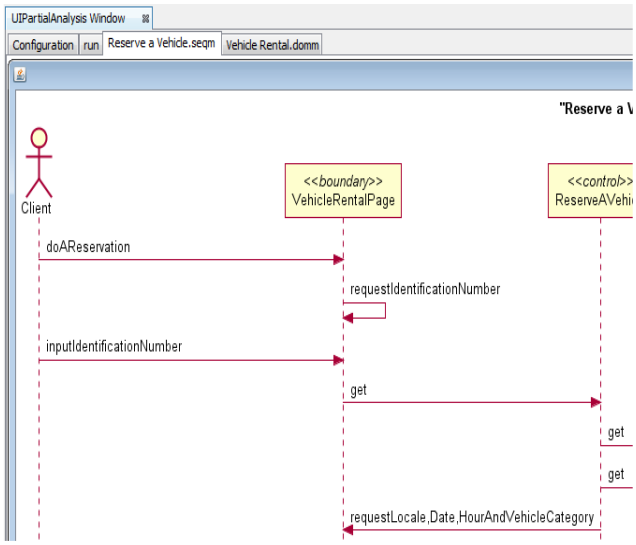

Figure 5. Sequence Diagram tab (partial view).

B. *PlantUML component and configurations*

The tool uses a pre-existing component known as PlantUML that permits generation of diagrams from stored commands in text formats. The tool creates PlantUML command files for each Sequence Diagrams to be generated from marked-up Use Cases and update with operations the PlantUML file related to the Domain Model. For this, the tool handles files with the following extensions:

- Files with “ucs” extension: File to be read and that contains a marked-up Use Case.
- File with “domm” extension: File to be read and updated and that contains PlantUML commands for Domain Model diagram generation.
- File with “seqm” extension: File to be created and that contains PlantUML commands for Sequence Diagram generation corresponding to Use Case with the same file name.

As soon as one file with PlantUML commands is generated or updated by the tool, immediately, the PlantUML component is activated for creating the respective diagram in the “png” format.

The tool, when in execution, alerts the analyst on cases of unidentified classes in the Domain Model, which however do not hinder generation of diagrams. The tool also alerts identified operations in Use Cases that already exists in the Domain Model. In this case, the tool does not include the operation in the Domain Model again.

The tool also transforms the names of objects in a manner by which words that compose it have their initials unified and transformed into capital letters. This pattern is known as “Camel Case”.

C. *Experiments*

The experiments were designed to verify if the application of the markups, associated with an automated method, could generate sequence diagrams with a reasonable margin of correctness, so that the adjustments to be made on

the model, after its generation, would not cost more than it would if the Analysis Model was made from scratch.

For the experiments, one looked for materials containing Use Cases with the respective Sequence Diagram and Class Diagram or Domain Models. The first material is a tutorial [21] about analysis with Use Cases. The second material is a training example [22] about Analysis Model.

One problem found during the experiments is that the Use Cases of both materials did not explicitly specify the interfaces, and this would lead to a poor initial Analysis Model. To try and solve this problem, we define the interfaces in the steps during the markup process.

The evaluation of the results was done by looking for missing messages and objects in the diagrams generated by the tool/method (generated diagrams) compared to the diagrams presented in the materials (original diagrams). During the evaluation, other types of differences are detected, and they are listed in the Table II with their respective quantity of occurrences. One important difference that occurs in both experiments is that an operation, in the control object, gets an inadequate name when the focus is on an interface and a system realizes two or more subsequent steps. In this case, the name given to the operation, in the control object, is the name related to the first step of subsequent steps, and then, it does not reflect the meaning of all messages involved. This type of problem should be corrected after the diagram is generated, because the method/tool does not possess the mechanism to label, in an adequate manner and in this situation, control object operations.

One threat to external validity is about the skill of the analysts to specify the Use Cases considering the markups. This job must be done in a manner that the Use Cases represent, as complete as possible, all important objects and interfaces that must be present in the partial Sequence Diagrams. If this does not occur, the generated diagrams will be poor. We consider that it is an important concern and it can be mitigated by training the analysts on the markups elements orienting them to try to use the markups as much as possible on the Use Cases. A future research could evaluate this supposition accordingly.

Other threat to external validity is about the lack of Domain Model during the Use Cases specification, possibly leading to ambiguous objects while writing Use Cases and precluding part of the method (the update of the Domain Model with operations). We consider that the method need to have their use restricted to cases where de Domain Model is available during the requirements specification. One future research could evaluate the impact of the absence of the Domain Model during the Use Case specification using this method.

TABLE II. TYPES OF DIFFERENCES BETWEEN ORIGINAL DIAGRAMS AND GENERATED DIAGRAMS BY THE TOOL AND QUANTITY OF OCCURRENCES IN EXPERIMENTS

Type	*Description of difference*	*Exp.#1*	*Exp.#2*
1	The behavior allocation (the object where an operation is placed) in the original diagram is different from what was explicited in the step, and so, is different from the alocation in the generated diagram. This difference configures a modeling decision of the analyst and cannot be inferred by the tool, so needs to be adjusted in the generated diagram after generation.	3	0
2	There are behavior details in original diagrams that does not appear in the generated diagram. This difference is acceptable because is part of an analysis work to go beyond the interaction between the actor and the System and the generated diagram is only partial.	3	1
3	Non-utilization, in original diagrams, of the boundary-control-entity pattern. This difference is acceptable because the tool applies this flow pattern and the difference does not necessarily configure mistake.	1	1
4	Message omission in original diagrams. This difference is not a problem but a omission of the analyst in the original diagram.	0	1
5	Inadequate name of an operation in crontrol object. This occurs when the focus is on an interface and the System perform two or more operations. In this case, the first message will give the name of the operation in the control object, but it will not reflect the meaning of the operation that does more things than the first message suggest.	1	1

Considering only the types of differences that deserve adjustments (types 1 and 5), we have 5 differences in both generated diagrams compared to the original diagrams. Considering yet that the two generated diagrams have 30 messages, there is 83 percent of similarities between original and generated diagrams. Therefore, generated Analysis Model should be revised by the analyst after generation for behavior allocations and for operation's name on the control object. Beyond this, the generated Analysis Model should be complemented, given that the generation is only partial and because of the limitations of the method, cited above.

V. CONCLUSION

This paper presented a set of markups for Use Cases and a transformation procedure for automatic partial generation of an Analysis Model. The Mason and Supsrisupachai [20] work was the basis for this work once it defined some markups for primitives in a Use Case in order to originate a Sequence Diagram. This work defines some more markups (markups for guard-condition, actors and interface) and defines a procedure to create a partial Sequence Diagram

 ISBN: 978-1-61208-449-7

with Analysis Classes, as well as complementing the Domain Model with the operations identified during the method execution.

The generated Analysis Model is composed of a partial Class Diagram and partial Sequence Diagrams (one per Use Case). The Class Diagram is the pre-existing Domain Model updated with the operations identified during the Sequence diagrams generation.

A tool was constructed based on the set of markups and the procedure in order to automate the generation of a partial Analysis Model.

The realized experiment demonstrated that the method/tool generates partial models with 83% of correctness, excluding differences that are not worth of adjustment. Considering this percentage, we believe that the implemented method could be used as a starting point for the Analysis Model since some improvements of the proposal can be made, as suggested below.

As a proposal to improve the tool, the model could be generated in XMI format, in a way that could be opened in a UML tool.

Another proposal to improve the tool is the creation of a tab for writing the marked-up Use Cases with an option for presenting texts with or without the markups, facilitating reading Use Cases when markups are hidden.

As a suggestion for future research, the procedure and the set of markups could consider alternative scenarios which will be transformed into fragments in the Sequence Diagrams. Extensions and Inclusions of Use Cases could also be considered.

Also, as a suggestion for future research, the transformation procedure and the set of markups could be extended to consider business rules written in Use Cases. A rule could be incorporated as an operation description when associated to a specific step or be incorporated as a note in the generated diagram when associated with Use Case as a whole.

REFERENCES

[1] J. T. Grose, G. D. Doney, and A. A. Brodsky, “Model Driven Architecture (MDA) and XMI,” in Mastering XMI. [S.l]: John Wiley & Sons, 2002, p. 329.

[2] D. Rosenberg and M. Stephens, “Introduction to ICONIX Process,” in Use Case Driven Object Modeling with UML: Theory and Practice. [S.l.]: Apress, 2007.

[3] I. Sommerville, Software Engineering, 9th ed.. [S.l.]: Addison-Wesley, 2011, p. 108.

[4] G. Booch, R. A. Maksimchuk, M. W. Engle, B. J. Young, J. Conallen and K. A. Houston, Object-Oriented Analysis and Design with Applications, 3th ed.. Boston, MA: Addison-Wesley, 2007, p.274.

[5] R. Pressman, “Requirements Engineering (RE) Tasks,” in Software Engineering: A Practitioner’s Approach, 6th ed.. [S.l.]: McGraw-Hill, 2004, p. 118.

[6] C. Larman, Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Iterative Development, 3th ed.. Upper Saddle River, NJ: Addison-Wesley, 2004, p. 145-146.

[7] IEEE. “Software Design,” in Guide to the Software Engineering Body of Knowledge. Los Alamitos, CA: [S.n.], 2004, p. 53.

[8] I. Jacobson, M. Christerson, P. Jonsson and G. Overgaard, Object-Oriented Software Engineering: A Use Case Driven Approach, 1st ed.. Edimburgh, UK: Addison-Wesley, 1992.

[9] B. Bruegge and A. H. Dutoit, “Analysis,” in Object-Oriented Software Engineering Using UML, Patterns and Java. Upper Saddle River, NJ: Pearson Prentice Hall, 2004, p. 177.

[10] G. Heineman and J. Denham, “Entity, Boudary, Control as Modularity Force Multiplier,” in Proc. 3rd Workshop on Assessment of Contemporary Modularization Techniques (ACoM.09), Orlando, FL, 2009, p. 42-47.

[11] L. Favre, Model Driven Architecture for Reverse Engineering Technologies: Strategic Directions and System Evolution, 1st ed.. Hershey, PA: IGI Global, 2010.

[12] D. Milicev, Model-Driven Development with Executable UML, 1st ed.. Indianapolis, IN: Wiley Publishing, 2009.

[13] N. Debnath, M. C. Leonard, M. V. Mauco, G. Montejano and D. Riesco, “Improving Model Driven Architecture with Requirements Models,” in Proc. 5th International Conference on Information Technology: New Generations (ITNG 2008), Las Vegas, NV, 2008, p. 21-26.

[14] T. Yue, L. C. Briand and Y. Labiche, “A Use Case Modeling Approache to Facilitate the Transition Towards Analysis Model: Concepts and Empirical Evaluation,” in Proc. Model Driven Engineering Languages and Systems (MoDELS 2009), Denver, CO, 2009, p. 484-498.

[15] T. Yue, L. C. Briand and Y. Labiche, “Automatically Deriving a UML Analysis Model from a Use Case Model,” Simula Research Laboratory, Oslo, Norway, Tech. Rep. 2010-15, Oct. 2010.

[16] L. Liwu, “A Semi-Automatic Approach to Translating Use Cases to Sequence Diagrams,” in Proc. Technology of Object-Oriented Languages and Systems (TOOLS’99), Nancy, France, 1999, p. 184-193.

[17] L. Liwu, “Translating Use Cases to Sequence Diagrams,” in Proc. 15th IEEE Int. Conf. on Automated Software Engineering (ASE’00), Grenoble, France, 2000, p. 293-296.

[18] L. Mendez, R. Romero and Y. P. Herrara, “UML Sequence Diagram Generator System from Use Case Description Using Natural Language,” in Proc. 4th Electronics, Robotics and Automotive Mechanics Conf. (CERMA’07), Cuernavaca, Mexico, 2007, p. 360-363.

[19] J. S. Thakur, A. Gupta, “Automatic Generation of Sequence Diagram from Use Case Specification,” in Proc. 7th India Software Engineering Conference (ISEC '14), 2014, Chennai, India.

[20] P. A. J. Mason and S. Supsrisupachai, “Paraphrasing use case descriptions and Sequence Diagrams: An approach with tool support,” in Proc. 6th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON 2009), 2009, Pataya, Thailand, p. 722-725.

[21] G. Evans. “Getting from use cases to code, Part-1: Use Case Analysis.” Internet: http://www.ibm.com/developerworks/rational/library/content/RationalEdge/jul04/TheRationalEdge_July2004.pdf, Jul. 13, 2004 [Feb. 22, 2015].

[22] J. White. “The Forgotten Step – Use Case Realization.” Internet: http://www.intertech.com/Blog/post/The-Forgotten-Step-Use-Case-Realization.aspx, Jan. 25, 2010 [Feb. 22, 2015].

[23] F. C. Souza. “Geração Automática de Diagramas de Sequência e Atualização do Modelo de Domínio a partir de Casos de Uso.” M.S. thesis, Institute for Technological Research, São Paulo, Brazil, 2011.

 ISBN: 978-1-61208-449-7