A Formal Semantics for Use Case Diagram Via Event-B

Thiago C. de Sousa[1*], Luciano Kelvin [2], Constantino Dias Neto[1], Carlos Giovanni N. de Carvalho[1]
[1] Technology and Urbanism Center, State University of Piauí, Teresina, 64003-750, Brazil.
[2] Institute of Mathematics and Statistics, University of São Paulo, São Paulo, 05508-090, Brazil.

* Corresponding author. Tel.: +55 86 3213 2424; email: thiago@uespi.br
Manuscript submitted December 2, 2016; accepted February 27, 2017.
doi: 10.17706/jsw.12.3.189-200

Abstract: UML has become the "de facto" standard for modeling object-oriented software. However, the UML notation suffers from an imprecise and incomplete semantics definition, which makes difficult to automated analysis and is errors-prone. Formal methods have been used largely in order to deal with this problem. This paper proposes an approach to formally describe Use Case Diagram using the Event-B language. More specifically, it is presented a set of transformation rules that maps the elements of this diagram to elements of Event-B, both expressed via meta-models. The approach is illustrated by showing an example about how to make automatic verification in this diagram.

Key words: Formal semantics, use case diagram, event-B, formal verification.

1. Introduction

The practice of modeling in a software development process is very important for understanding the requirements. Partial models are created to represent the structural and behavioral aspects of the system, so the developers can work at distinct levels of abstraction before start the codification. Another important factor in a software development process is the verification for inconsistencies during its early stages, which means that the entire elaborated model should be consistent with itself refinement, with others models and with global constraints (business rules and properties).

Generally, the models are designed using the Unified Modeling Language (UML) [1], which provides some diagrams to describe a system from different perspectives. This language is very expressive and easy to learn, however imprecise and ambiguous. On the other hand, formal methods, through mathematical techniques, allow specifying requirements of a system, in order to maintain consistency, completeness and absence of ambiguity. Moreover, formal methods provide a lot of tools for automatic verification of inconsistencies.

This work presents an approach for formalizing Use Case Diagram (UCD) using the Event-B formal method [2]. More precisely, it is provided a mapping for each element of this diagram to one or more elements of Event-B, and showed how the proposal is useful for supporting a formal semantics to UCD and a mechanism to check for inconsistencies.

The next section presents the related work. After that, it is explained some theoretical fundamentals, introducing the main concepts of the Event-B language, as well as the elements of Use Case Diagram. The section 4 shows the transformation rules from UCD to Event-B. An example of inconsistency verification is presented in section 5. Finally it is reserved the last section for further discussions about the proposed formalization and directions for future work.

2. Related Work

As previous mentioned, UML diagrams have become increasingly common to describe the software specifications, because of its graphical notation, which facilitates the understanding of the software model. However, for these simplicities, the diagrams may contain errors, inconsistencies and ambiguities.

There are some works that propose the translation of Use Case Diagram to a formal language. In [3] there is the translation of UCD to the Z formal language. In [4] is presented the mapping of the UCD properties to CASL (Common Algebraic Specification Language). In [5] it is showed a research for identifying inconsistencies in UCD using ASML (Abstract State Machine Language). However, these formal languages have poor tool support and are not widely accepted by the industry.

There are some works that propose the translation of UML diagrams to B/Event-B in order to utilize the expressive power and the tool support of these notations, which are two of the widely used formal methods by the industry. In [6] the authors propose a mapping of class diagram with OCL (Object Constraint Language) rules to the B language. In [7] there are some rules to translate state and collaboration diagrams to B. In [8] is presented a graphical tool to automatically translated class and state diagrams to Event-B. These works show the importance of presenting a formal semantics to UML diagrams as our proposal does.

There are also some works that suggest the mapping from Use Case Specifications (UCS) to B/Event-B. In [9] is proposed a controlled natural language (CNL) to UCS in order to facilitate de translation of transactions as B operations. In [10] is presented a tool to generate Event-B models from informal UCS using some formal counterparts, i.e. pre- and post-conditions along with triggers and actions associated with the various flows. However, these works only focus on the textual use case specification, which is a part of the Use Case Model. The other part is the Use Case Diagram, which is also widely used by the industry and is the core of this proposal. So, in this work is showed an approach to translate the Use Case Diagram, one of the most useful UML diagram, to Event-B, one of the widely used formal language.

3. Background

3.1. Event-B

Event-B is a state model-based formal method for modeling systems based on predicate logic and set theory where the refinement mechanism, which allow to build a model gradually by making it more and more precise (that is, from an abstract model to a concrete one), and the consistency checking, which permit to verify the validity of the properties of a system, are guaranteed by mathematical proof obligations. These features have been supported by an open-source platform (based on Eclipse IDE) named Rodin [11] that is constantly improved and extended by the community via plugins.

The Event-B language has two components: contexts and machines. Machines and contexts may have several relationships: a machine refines another and a context extends another one. Furthermore, a machine sees one or more contexts. A context is used for static specification, including global sets, constants and axioms and its structure is [context C1 extends C2 sets S constants C axioms A end], where C1 and C2 are context names, S and C are list of set and constant, respectively, and A represents axiom rules. A machine is applied to dynamic specification, containing variables, invariants and events and is described usually by [machine M1 refines M2 sees C1 variables V invariants I events E end], where M1 and M2 are machine names, C1 is a context and V, I and E are list of state variables, invariants and events respectively.

An event is composed by two elements: guards and its parameters, which are valid conditions for the occurrence of the event; and actions, which denote how the variables change. In general, an event has the form [any x where G then A end], where x is a list of parameters, G is a list of guards and A is a list of actions. But, it is also possible to have an event construction without parameters, becoming the form [when G then A end]. In Listing 1 we have an example of an Event-B model.

MACHINE
DoctorsOffice
SEES
DoctorsOffice_ctx
VARIABLES
control_MakeAppointment
INVARIANTS
Inv1 : control_ MakeAppointment ∈ Status
EVENTS
INITIALISATION ≙
BEGIN
act1 : control_ MakeAppointment := nostatus
END
MakeAppointment ≙
WHEN
grd1 : control_ MakeAppointment := started
THEN
act1 : control_ MakeAppointment := ended
END
END

CONTEXT
DoctorsOffice_ctx
SETS
Status
CONSTANTS
nostatus
started
ended
waitingforMakeAppointment
AXIOMS
axm1 : partition(Status, {nostatus}, {started}, {ended}, {waitingforMakeAppointment})
END

Listing 1. Example of event-B model.

3.2. Use Case Diagram

The Use Case Diagram is one of the diagrams that make up the UML (Unified Modeling Language), serving to identify use cases, a set of sequences of actions that one system can run in a scenario, and actors, which represent an external factor (a user or an external system) that interacts with the system. Sometimes to describe a system it is necessary to modularize the behaviors and the software scenarios, and therefore a use case can extend and/or include other use cases. In Fig. 1 we have an example of a Use Case Diagram with these components.

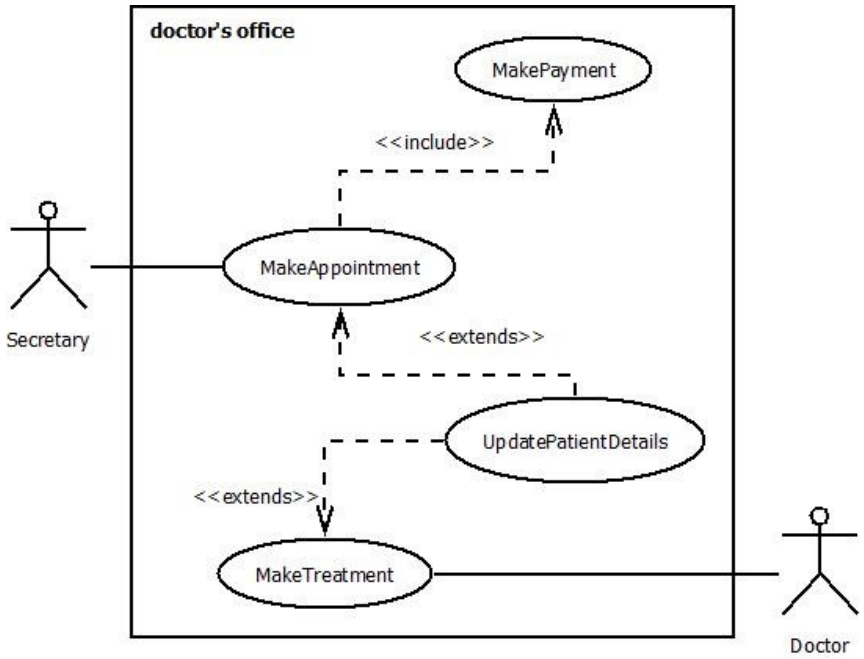

Fig. 1. Example of the Use Case Diagram Components.

4. From Use Case Diagram to Event-B

For the purpose of the Use Case Diagram formalization we have used the transformational semantic approach [12], which maps a language *L1* into another language *L2*, which already has semantics, in order to assign semantics to *L1*.

Practically, we used the Eclipse Modeling Framework (EMF) [13], a modeling framework that enables developers to quickly build tools and other robust applications based on meta-models and, from these, create different codes. In EMF, from the point of view of the transformational approach, the language that it wants to assign semantics and the language that already has a well-defined semantics are integrated on a common basis in the Meta Object Facility - MOF [14], which is the OMG standard meta-language for UML and other modeling languages.

Given two meta-models of different notations, a mapping between them is defined via a set of transformation rules expressed by QVT (Query / View / Transformation) [15], which is the OMG standard transformation language. The Query should get a model as input and select some specific elements. The View refers to models derived from other models as the result of a query, and the Transformation is associated with receiving a given model as an input parameter and its updates, or related to the creation of another model as output.

In the next sub-section it is presented the Event-B language and Use Case Diagram meta-models. After that, the transformation rules between them are showed. In this case, to facilitate understanding and avoid a heavy explanation of the QVT notation, the authors decided to provide the suggested mapping by examples.

4.1. Meta-Models

The Event-B language has officially a meta-model named EMF Event-B [16], in order to provide integration with other tools that use EMF technology. The meta-model is divided into three elements: the core package and two sub-packages, one for the Machine component and other for the Context component. Again, for readability, it is presented only the sub-packages.

In Fig. 2 it is possible to visualize the Context sub-package of the Event-B meta-model. This sub-package shows that a context can extend other contexts and can be composed by global sets, constants and axioms.

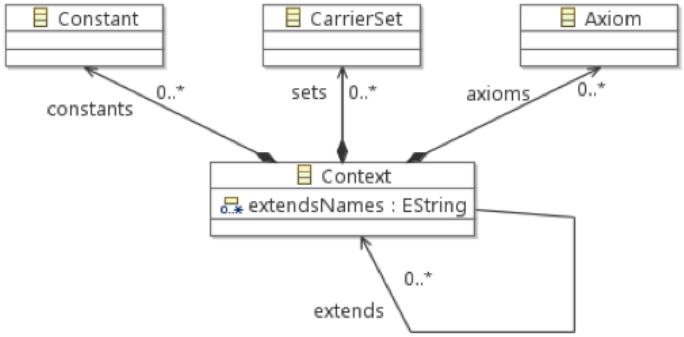

Fig. 2. Event-B Context Meta-model.

In Fig. 3 it is showed the Machine sub-package of Event-B meta-model. This sub-package contains the elements that may represent an Event-B machine. It can be observed that a machine can refine another machine, see contexts, and may have variables, invariants, variants and events. Each event can refine another event and can contain parameters, guards, witnesses and actions.

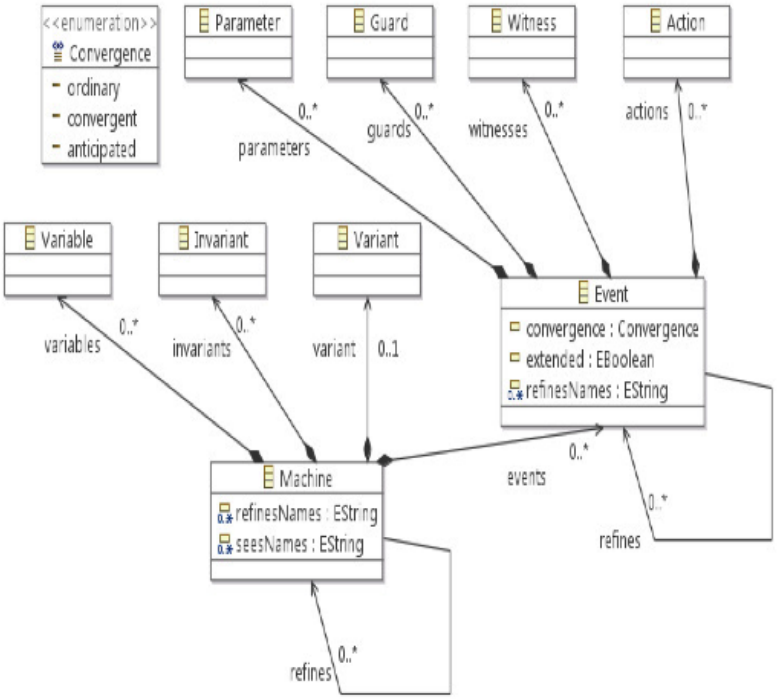

Fig. 3. Event-B Machine Meta-model.

The Use Case Diagram meta-model defines the elements that initially represent the software behavior. The proposed approach is for maintaining only the essential components for modeling this diagram, creating a basic meta-model and leaving some restrictions to be defined externally, as advocated by [17]. Thus, the presented meta-model is simpler than that specified by the OMG to Use Case Diagram, with only the most used elements, namely: actors, use cases, and the links between them. In addition, it was added two specialized meta-class in order to represent *include* and *extend* stereotypes. In Fig. 4 it showed the proposed meta-model for the Use Case Diagram.

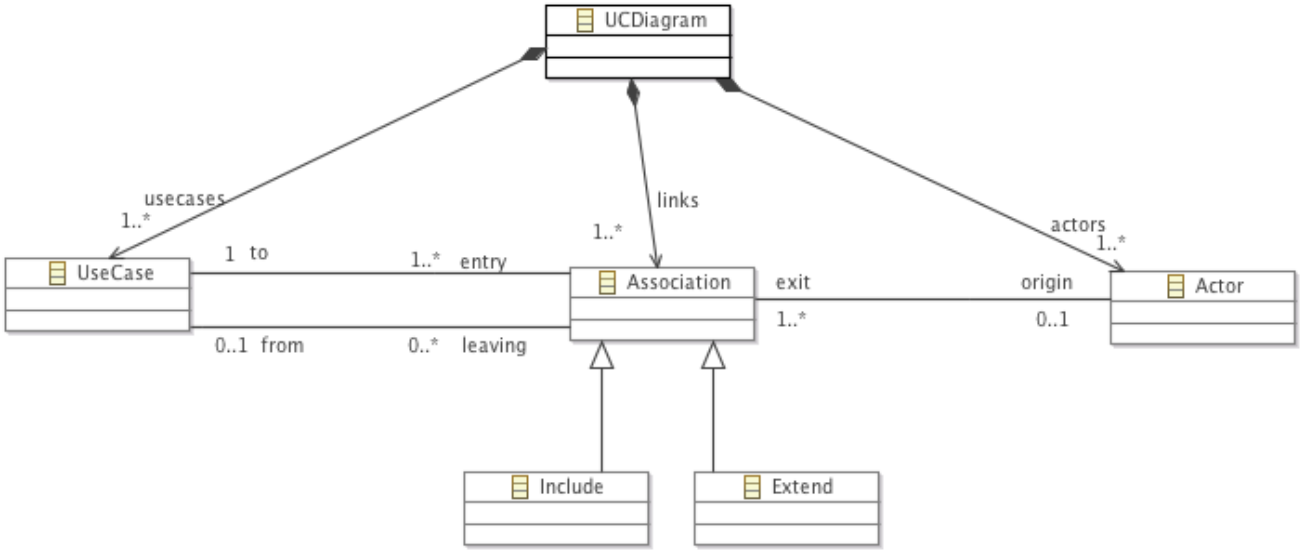

Fig. 4. Use Case Diagram Meta-model.

The specialization between use cases is not part of the meta-model because it is not possible to define via diagram if a use case is abstract or concrete, and there is no mention in UML 2.5.x about it. So, the relations between use cases are made only using i*nclude* and *extend* stereotypes. The multiplicity between an actor and a use case was also not included due to the vagueness semantics of UML, because the interaction between instances of actors/use cases can be simultaneous (concurrent), or overlapping, at different points in time, or mutually exclusive (sequential, random, etc.). The generalization between actors was also absent as it is possible to semantically represent it directly connecting the concrete actors to use cases connected with the abstract actors. Finally, it was decided to remove the extension point, since their semantics depend on the reference position in the use case textual specification, which cannot be done by diagram.

4.2. Transformation Rules

In this subsection, for each one of the Use Case Diagram elements, it will be presented its equivalent in the Event-B language. First, it is important to interpret what means a Use Case Diagram instance. In the proposal its name is mapped as a Machine name in Event-B. Moreover, an empty INITIALISATION event is created in the Machine. In the Context, it generates some elements: a CarrierSet named “Status”; three Constant instances named “nostatus”, “started” e “ended”; and an Axiom that enumerates (partitions) the constants as possible states of “Status”.

Another component that generates elements in Event-B is a Use Case instance. A Constant instance named “waitingfor{Use Case name}”, that is included in the “Status” partition, and a set denominated "{Use Case name}_SET" are created in the Context. In the Machine, some instances are generated: a Variable named “control_{Use Case name}”; an Invariant, indicating that the variable has a status, named “control_{Use Case name} : Status”; an Action into the INITIALISATION event, denominated “control_{Use Case name} := nostatus”, indicating the initial lack of status; and an Event, standing for the completion of the use case, named “{Use Case name}”, with “control_{Use Case name}=started” as Guard and “control_{Use Case name} := ended” as Action.

The mapping of an Actor instance is similar to the Use Case mapping. It generates a Variable named “control_{Actor name}”; an Invariant, indicating that the variable has a status, named “control_{Actor name} : Status”; an Action into the INITIALISATION event, denominated “control_{Actor name} := nostatus”, indicating the initial lack of status; and two Events: the first one to start the actor instance, named "{Actor name}starts", with "control_{Actor name}=nostatus” as Guard and “control_{Actor name} := started” as Action; and the second one to finish the actor instance, named "{Actor name}", with "control_{Actor name}=started” as Guard and “control_{Actor name} := ended” as Action.

An Association instance is used for the invocation of a Use Case by an Actor and is directly mapping to an Event named "{Actor name}invokes{Use Case name}", with "control_{Actor name}=started" and "control_{Use Case name}=nostatus” as Guards and “control_{Use Case name} := started” as Action.

An instance of Extend meta-class can generate different instances in an Event-B Machine, depending on the modeled relationships. It creates an Event named "{Use Case to name}extendedby{Use Case from name}" for each extend connection between two use cases. In this event, it creates two Guards ("control_{Use Case to name} = started" and "control_{Use Case from name} = nostatus") and two Actions ("control_ {Use Case to name} := waitingfor{Use Case from name}" and "control_ {Use Case from name} := started"), indicating the status necessary to extend a Use Case and how its remains after the initialization. If a Use Case is extended by two or more Use Cases, then in the "control_ {Use Case to name} = started" Guard of the generated Event is introduced a disjunction "control_ {Use Case to name} = waitingfor{Use Case from name) ^ control_{Use Case from name} = ended" for each of the other related Use Cases, indicating that the extension may occur, or when the extended Use Case is initialized (started), or when one of the other

related Use Cases are finished (ended). Moreover, for each related Use Case, a disjunction "control_ {Use Case to name} = waitingfor{Use Case from name} ^ control_ {Use Case from name} = ended" is also added to the "control_ {Use Case to name} = started" Guard of the Event generated by the Use Case instance, indicating that it can be finalized, or without be extended by any other, or after the extension of any of the related Use Cases.

Finally, the mapping of an Include instance to Event-B is analogous to the Extend mapping, with some particularities related to controlling the number of included Use Cases. There is a Variable named "control_{Use Case from name}_include"; an Invariant, indicating the integer typing of the variable, named "control_{Use Case from name}_include : N"; an Action into the INITIALISATION event, denominated "control_{Use Case from name}_include := 0", indicating that no inclusion Use Case was used yet. The other differences are: instead of generates an Event named "{Use Case to name}extendedby{Use Case from name}" is produced an Event called "{Use Case from name}includes{Use Case to name}", but the Guards and the Actions are equal; there is a new Action in the generated Event named "control_{Use Case from name}_include := control_{Use Case from name}_include + 1", which means the invocation of one more included Use Case; and "control_{Use Case from name}_include = X" is also added to the Guards of the Event generated by the Use Case instance, where 'X' is the number of included Use Cases, representing that it can be finalized only after the completion of all included Use Cases.

In Table 1 it is provided an incremental summary using the example presented in the Fig. 1 in order to facilitate the visualization of the translation from Use Case Diagram to Event-B.

Table 1. UCD Mapping to Event-B

Use Case Diagram	**Event-B**	**Example (partial)**
Diagram	Sets	Status
	Constants	nostatus, started, ended
	Axiom	axm1 : partition(Status, {nostatus}, {started}, {ended})
Use Case	SET	MakeAppointment_SET
	Constants	waitingforMakeAppointment
	Variable	control_MakeAppointment
	Invariant	inv1: control_MakeAppointment ∈ Status
	INITIALIZATION	act1: control_MakeAppointment := nostatus
	Event	EVENT MakeAppointment ≙ WHEN grd1: control_MakeAppointment = started THEN act1: control_MakeAppointment:= ended END
	Axiom	axm1: partition{Status, {nostatus},{started},{ended},{waitingforMakeAppointment}}

Actor	Variable	control_Secretary
	Invariant	control_Secretary ∈ Status
	INITIALIZATION	act2: control_Secretary := nostatus
	Event	EVENT SecretaryStarts ≙ WHEN grd1: control_Secretary = nostatus THEN act1: control_Secretary:= started END EVENT Secretary ≙ WHEN grd1: control_Secretary = started THEN act1: control_Secretary:= ended END
Association	Event	EVENT SecretaryinvokesMakeAppointment ≙ WHEN grd1: control_Secretary = started grd2: control_MakeAppointment = nostatus THEN act1: control_MakeAppointment = started END
Extend	Event	EVENT MakeAppointmentextendedbyUpdatePatientDetails ≙ WHEN grd1: control_MakeAppointment = started grd2: control_UpdatePatientDetails = nostatus THEN act1: control_MakeAppointment = waitingforUpdatePatientDetails act2: control_UpdatePatientDetails = started END
Include	Variable	control_MakeAppointment_include
	Invariant	inv2: control_MakeAppointment_include ∈ N
	INITIALIZATION	act3: control_MakeAppointment_include := 0
	Event	EVENT MakeAppointmentincludesMakePayment ≙ WHEN grd1: control_MakeAppointment = started grd2: control_MakePayment = nostatus THEN act1: control_MakeAppointment = waitingforMakePayment act2: control_MakePayment = started act3: control_MakeAppointment_include:= control_MakeAppointment_include + 1 END EVENT MakeAppointment ≙ WHEN grd2: control_MakeAppointment_include = 1 THEN END

5. Use Case Formal Verification

In the previous section it was presented the proposal to formalize the Use Case Diagram into Event-B language, showing the meta-models of these notations, as well as the transformation rules to perform this procedure. However, it was not expounded how the formalization of the Use Case Diagram can be used to avoid possible modeling inconsistencies issues. In this section it is showed an example of the power of Event-B in order to help in this mission. Suppose that an inexperienced UML user has designed the Use Case Diagram presented in Fig. 4.

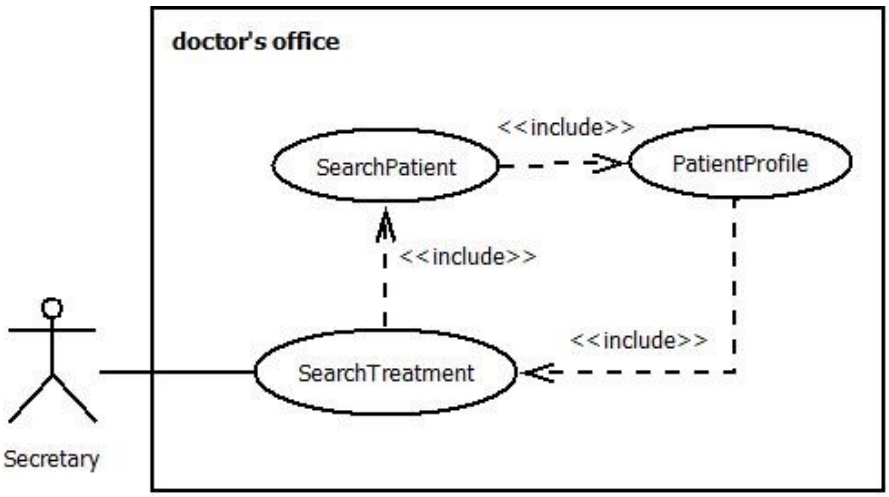

Fig. 4. Use Case Diagram with Inconsistencies.

This is one of the classical Use Case Diagram semantics problems and, according the best modeling practices, should be avoided, since the software can reach a deadlock state. In other words, this restriction implies that any hierarchy of the *include* stereotype must be acyclic. In order to provide automatic verification of this issue it must use the axioms and theorems constructions of Event-B.

In the Event-B Context it must add a constant named "const_{Use Case name}" for each instance of a Use Case in the diagram and, for each of these, introduces an axiom denominated "const_ {Use Case name} : OBJECT_SET", indicating it is part of the global set representing the possible existing instances. It must also add a constant named "inc" to represent the inclusion relations between instances of Use Cases, with an axiom denominated "inc = {const_{Use Case from name} -> const_ {Use Case to name}, ... }". In order to perform this verification, a theorem for cycle detection ($\neg (\exists z.\ z \subseteq dom(inc) \ \wedge \ \emptyset \subset z \ \wedge \ z \subseteq inc[z])$) must be included . So, the B-Event model creates automatically a proof obligation for this theorem that can be checked by the Rodin platform.

The Event-B Context inclusions can be viewed in the Listing 2.

CONTEXT
DoctorsOffice_ctx
SETS
OBJECT_SET
CONTANTS
inc
const_SearchPatient
const_PatientProfile
const_SearchTreatment

AXIOMS

axm1 : const_SearchPatient ∈ OBJECT_SET

axm2 : const_PatientProfile ∈ OBJECT_SET

axm3 : const_SearchTreatment ∈ OBJECT_SET

axm4 : inc = { const_SearchPatient ↦ const_PatientProfile, const_PatientProfile ↦ const_SearchTreatment, const_SearchTreatment ↦ const_SearchPatient}

thm1 : ¬ (∃z · z ⊆ dom(inc) ∧ ∅ ⊂ z ∧ z ⊆ inc[z])

END

Listing 2. Event-B Context generated for the example of Fig. 4.

And the reported error presented by the Rodin Platform is showed in Fig. 5

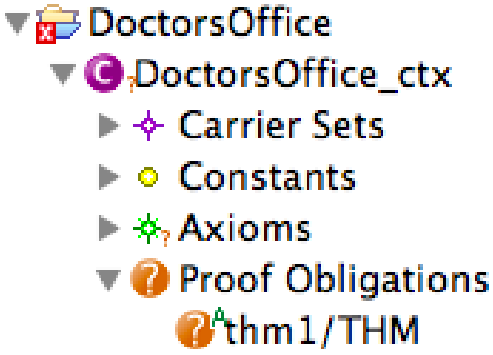

Fig. 5. Pop up error presented by the Rodin Platform

It is important to emphasize that this example is used only to show how the formalization power provided by Event-B can be used to detect inconsistencies in the Use Case Diagram modeling. Many other Use Case Diagram syntax and semantic issues can be checked, injecting axioms and theorems to be proved (most automatically) by the Rodin platform.

6. Discussions and Future Work

UML has become the "de facto" standard for modeling software, providing an easy and universal notation. However, UML suffers from an imprecise semantics, which makes difficult to automated analysis. On the other hand formal methods have been used largely in order to deal with this issue, but is not too accepted as UML as a modeling language. So, there are some studies trying to merge UML with formal methods.

Although the Use Case Diagram is one of the most used UML diagrams by the industry and the absence of formal semantics to this diagram is showed a very important issue to software development, few studies have been carried out in this direction, i.e. providing a formal semantics to the Use Case Diagram. Moreover, usually, these studies propose the translation to formal languages that neither have not good tool support nor are widely accepted by the industry.

This paper has proposed an approach for mapping Use Case Diagram into Event-B, providing a contribution towards a mechanism to make automatic verification in this diagram, which may bring the benefits of formal methods to industrial practitioners. In particular, the proposal provides the translation of the main UML Use Case Diagram elements (actor, use case, include and extend) to an Event-B model such that the proof the obligations generated during the process can be proved automatically by the Rodin platform and so the user can have a feedback in order to avoid an inconsistency diagram issue.

This work must have some important future experiments. One of them is the inclusion of other use case diagram elements such as generalization, multiplicity, etc. Another improvement is the extension of the approach to incorporate formal use case descriptions, so it is possible to use the extension point in the diagram. It is also possible to smooth the mapping from Event-B to UML of the detected errors in order to get more understandable feedbacks. Finally, it can also be interesting to provide a more complete set of inconsistency rules in order to have a more useful and robust feedback set.

References

[1] Rumbaugh, J., Jacobson, I., & Booch, G. (2004). Unified modeling language reference manual. *The Pearson Higher Education.*

[2] Abrial, J.-R. (2003). Event based sequential program development: Application to constructing a pointer program. *Formal Methods.*

[3] Sengupta, S., & Bhattacharya, S. (2006). Formalization of UML use case diagram - A Z notation based approach. *Proceedings of the 2006 International Conference on Computing and Informatics.*

[4] Mondal, B., Das, B., & Banerjee, P. (2014). Formal specification of UML use case diagram - A CASL based approach. *International Journal of Computer Science and Information Technologies*, *5(3)*, 2713-2717.

[5] Shen, W., & Liu, S. (2003, November). Formalization, testing and execution of a use case diagram. *Proceedings of the International Conference on Formal Engineering Methods* (pp. 68-85).

[6] Leading, H., & Souquieres, J. (2002, June). Integration of UML views using B notation. *Proceedings of Workshop on Integration and Transformation of UML Models*.

[7] Laleau, R. & Mammar, A. (2000). An overview of a method and its support tool for generating b specifications from UML Notations. *Proceedings of the IEEE International Conference on Automated Software Engineering*.

[8] Snook, C., & Butler, M. (2008). UML-B and event-b: An integration of languages and tools. *Proceedings of The IASTED - International Conference on Software Engineering*.

[9] De, S. T. C., & Russo, J, A. G. (2010). Starting b specifications from use cases. P*roceedings of Abstract State Machines*.

[10] Murali, R., Ireland, A., & Grov, G. (2016). UC-B: Use case modelling with event-b. *Proceedings of Abstract State Machines.*

[11] Abrial, J. R., Butler, M., Hallerstede, S., Hoang, T. S., Mehta, F., & Voisin, L. (2010). Rodin: An open toolset for modelling and reasoning in event-b. *Int. J. Softw. Tools Technol. Transfer*, *12(6)*, 447–466.

[12] Lano, K. (2009). *UML 2 Semantics and Applications*. John Wiley & Sons.

[13] Steinberg, D., Budinsky, F., & Paternostro, M. (2008). Merks, EMF: Eclipse Modeling Framework 2.0. Pearson Education, New Jersey, US.

[14] Group, O. M. (2011). MOF 2.4.1 Specification. 2011. Retrieved from: http://www.omg.org/spec/MOF/2.4.1

[15] Gronback, R. C. (2009). Eclipse modeling project: A domain-specific language (DSL) toolkit. Pearson Education.

[16] Snook, C., Fritz, F., & Illisaov, A. (2010). An EMF framework for event-B. *Workshop on Tool Building in Formal Methods*.

[17] Fondement, F., Muller, P. A., Thiry, L., Wittmann, B., & Forestier, G. (2013, September). Big metamodels are evil. *Proceedings of the* I*nternational Conference on Model Driven Engineering Languages and Systems* (pp. 138-153).

Thiago C. de Sousa is a assistant professor at the Technology and Urbanim Center of the State University of Piauí (UESPI) – Teresina, Brazil. He received the B. Sc. and the M.Sc. degrees in computer science from the University of São Paulo, Brazil, in 2002 and 2007, respectively, and the Ph. D. degree in Electrical Engineering from the University of São Paulo, Brazil, in 2013, with a doctoral training at the University of Southampton, England, in 2011. He has experience in formal methods and model checking.

Luciano Kelvin received the B. Sc. degree in computer science from the State University of Piauí, Brazil, in 2013 and the M. Sc. degree in computer science from the University of São Paulo, Brazil, 2016. He has worked as a developer in Claudino Group and Alstom / QI agency. He has experience in software engineering area, mainly in the following areas: formal methods, model verification and specification mining.

Constantino Dias Neto received the technologist degree in Data Processing from Association of Piaui Higher Education (1998) and the M. Sc. degree in Computer Science from the Federal University of Pernambuco (2003). Nowadays he is a PhD candidate in Biotechnology. He is currently Lecturer of the State University of Piauí and researcher at the ubiquitous systems laboratory and pervasive.

Carlos Giovanni N. de Carvalho is master in computer science from the Federal University of Pernambuco - UFPE (2003) and worked on computer networks, especially in protocols of data link layer and transport layer. He is also a doctor of engineering teleinformática from the Federal University of Ceará - UFC (2012) and worked on data reduction in wireless sensor networks. Currently he is a assistant professor of Computer Science and the director of the Center for Technological Innovation of the State University of Piauí.