**ABSTRACT**

This thesis describes a plugin which was created for developers who intend to design their models using the Event-B language. Event-B is a formal language which uses mathematical techniques for system modelling and verification. The accuracy of the model is ensured by proof obligations. The main disadvantage of the Event-B is that it doesn’t have many well-developed modularization constructs and it is not easy to combine specifications in Event-B with those written in other formal languages. Developers can use the plugin described in this thesis if they want to make changes in the existing model such as element renaming or merging, without writing new elements from the scratch. Developers can use their knowledge of the institution theory and specification based operators to interact with the plugin.

Software development requires from the developer not only an accurate and clear structuring of the system, as well as efficient tests for finding bugs, but also a strong mathematical proof. All these components allow software to be reliable and eliminate the possibility of system failure. The more complicated the system is, the more difficult it becomes to make sure that it works correctly and, in this context, mathematical proof can help to show the absence of bugs.

**CHAPTER 1**

**Introduction**

Developing dependable systems is one of the most important targets in software development nowadays. Technologies are involved in human’s daily life. People use smartphones, laptops, and different vehicles to solve their everyday problems. They rely on their gadgets which sometimes can work improperly and cause troubles. In case when the systems should be reliable and should work stable all the time, software, on which these systems are based, has to be verified. Verified software allows user to be sure that it will work accurately in any situation. Reliable software usually is very expensive and uses in situations when the software fault causes huge money losses or endanger people’s lives. To decrease the danger of money losses or secure people, scientists analyze all the possible variants of the developing system behavior and software developers design systems which work stable all the time. But analyzing the system is very time-consuming and difficult work, the developer should be concentrated all the time, that is sometimes impossible. To solve this problem and eliminate the human factor special tools were developed. They allow to verify the software easier and faster. One of these tools is Rodin Platform, which provides effective support for refinement and mathematical proof [http://www.event-b.org]. Event-B is not the only one language which can be used in verification, but the most popular one. Practical work of this thesis based on creating a Plug-in which manipulates with Event-B elements with the help of formal expressions. The Rodin Platform is Eclipse-based, so the Plug-in is written in Java. The thesis should help people who is familiar with theory of institutions and specification-based operators (SBO), but not familiar with Event-B syntax, simply manipulate with Event-B elements by using SBOs inside the Rodin Platform.

* 1. **Motivation**

During the last couple of decades, the popularity of the software increased vastly. No one can imagine the work of banks, schools, hospitals without using modern technologies. Almost every single household appliance works under control of the software. The timetable for public transport is not created manually more. People trust the software to control airplane and train traffic. The life became more convenient, but more dangerous at the same time. If the software development is quite difficult work, the reliable software development is much more difficult. It requires not only accurate and clear structure and efficient tests to find the bugs, but also strong mathematical proof. The software verification is very important, because the software testing can show the presence of the bugs, but never their absence [Dijcstra]. The software verification can be done using the formal methods. During the studying of formal methods, I met the problem that knowing only theory of institutions is not enough to start working with tools which provide software verification. One should not only understand the logic of the verification, but although study the syntax and logic of the verification language and tool he uses. Each platform has its own features and to study them all, person should spend lots of time. It is not often easy, because developer may want to try another language and another tool and should start studying formal language and verification tool from the very beginning. If there is a code written in some formal language, it becomes another problem to translate this code to another formal language. Event-B is quite popular language, but there are some known concerns regarding the using of Event-B formal language:

* it doesn’t have well-developed modularization constructs and it is not easy to combine specifications in Event-B with those written in other formalisms [Marie’s paper];
* Rodin Platform, which maintains the Event-B language is quite big and has lots of features, so it becomes quite difficult to start using it without knowing the Event-B;
* If the developer is familiar with the theory of institutions, but not familiar with the Event-B, it will be hard to start working with Rodin Platform.

This theme for the thesis was chosen to decrease developers’ efforts in writing dependable software and to make its creating as easy as possible. The best way of doing this is to make it possible to use the Rodin Platform to write some specification-based operators and manipulate with elements, written in Event-B. This will allow to save time and to speed up the software development. Developers, using the formal methods to proof their programs will receive the opportunity to get the flexible tool which can be used to write proof obligations not only in Event-B, but also with the help of specification-based operators and to manipulate with Event-B elements without knowing its syntax. The main currency nowadays is time, so the studying lots of formal languages and understanding the main features of the tools, which use these languages, require lots of time that can be spend on the better software design.

* 1. **Overall project objective**

This thesis will provide a solution for the problem, described in the previous part. The benefit for developers will be in the opportunity to use the Rodin Platform to make changes in Event-B models not only using the Event-B formal language, but also using the specification-based operators. One of the most important possibilities of the created plugin is the renaming Event-B model elements (e.g., machines, events, variables), as it becomes a huge problem, when the model is quite big and the renaming object is used in different places. Renaming also can cause problems in proof obligations, one of the specific parts of formal languages, so it was important to keep this part valid and avoid errors caused by the renaming. The second big part of the plugin is merging two machines together, combing their variables, events, etc. This feature will allow two machines to combine their properties into one machine without creating new machine from the scratch. Despite the fact that renaming (refactoring) and creating composition plugins already exist, the new plugin will combine these two features and will be run by specification-based operators. The developed plugin will simplify the verification of the software in formal languages and save time for the development of the reliable software systems.

**1.3 Research question**

The main question of the project is **RQ1**, *how can the software verification be simplified for the developers?* This question is divided in two separate research questions, RQ1.1 and RQ1.2.

* **RQ1**: how can the software verification be simplified for the developers?
* **RQ1.1**: how can developers, using different formal languages use the Rodin Platform without necessity of studying Event-B language?
* **RQ1.2**: how can renaming and machine composition be implemented without any errors in proof obligations?

These questions describe the main idea of the current project. Understanding of what is done in this field and studying the existing plugins and projects will be the first step in answering these questions. The existing plugins may help in designing the new one in more simple and efficient way. With a firm understanding of the work done in this research field, the work that should be done will become more precise. The resulting developed plugin will provide developers with short and easy to use tool which will allow to use the Event-B models without long studying and understanding all the Rodin Platform features.

**1.4 Solution**

To solve the above stated questions, the plugin for the Rodin Platform was designed and developed. It will simplify the development of the reliable software systems and help developers to start using the Rodin Platform simply and without huge efforts. The plugin will use the existing Event-B models and manipulate with its data. The plugin supports the input validation and indicates if some problems can occur during the operations (renaming or composition) execution. The plugin is developed in Java and uses the Event-B abstract syntax tree in the syntax analysis part of the development. This approach allows user to make necessary changes without the necessity of parsing the XML-files, which store the Event-B project. The plugin is addressed to the developers who wants to verify the software and who is familiar with the theory of the institutions and specification-based operators.

**1.5 Structure of the thesis**

The project is separated into six chapters:

*Chapter 1:* current chapter, introduces the initial problem, describes the motivation of the current project, states the research questions and solution of the problem in short. This part also describes the current problems of the using Rodin Platform.

*Chapter 2:* this chapter presents the background research, it names and describes in detail two plugins that are most closely related to the current project and shows is advantages and disadvantages. This information is used in the next chapter in comparison between this project and existing plugins.

*Chapter 3:* this part defines the key requirements for the project and describes the design and implementation phases of the development process.

*Chapter 4*: this chapter represents the evaluation the work and states if all the key requirements were met and includes the comparison with existing plugins, mentioned in chapter 2.

*Chapter 5*: this part describes the case study.

*Chapter 6*: the last chapter of the thesis contains conclusions, where the value of the work is described and the future possible work is outlined.

**CHAPTER 2**

**Background and related work**

The formal reasoning is very ‘strict’ type of reasoning, it helps to find answers and make decisions between the conflicting sentences, ideas or opinions of different people. The formal reasoning is based on the certain form of arguments, which are declared to be true. The arguments, which contradict these arguments, become false accordingly. The conclusion, which is based on true statements, supposed to be true as well. Formal reasoning usually uses reasons in terms of logic, not simple words. There are several languages, based on formal reasoning, exist. One of the most popular languages based on formal reasoning is Event-B. It was developed by  [Jean-Raymond Abrial](https://en.wikipedia.org/wiki/Jean-Raymond_Abrial) ([France](https://en.wikipedia.org/wiki/France)) and based on B language. The main difference between B and Event-B is that Event-B has simpler notation, it is easy to learn and use and it has more features. It is used in lots of industrial projects and allows to create systems and verify them.

**2.1 Event-B**

Event-B is a notation for formal modelling based around an abstract machine notation [Rodin User’s Handbook]. It allows the user to verify difficult real-life tasks. There are some examples of using the verification in daily life: smart-grid modelling and railway interlocking models. Verification of the systems is used to ensure the safety of people or to avoid costs caused by improper work of the system. The main advantage of using the Event-B is that all development errors in the model can be easily found since in incomplete and inaccurate model some proofs cannot be done.

**2.1.1 Contexts and machines**

Event-B models consist of two main parts: contexts and machines. Context shows all static parts of the model, while machine represents dynamic parts of the model. These two main parts allows creating efficient models and describing the behavior of the system. The key feature of the Event-B is that the primary model can be really simple, but with the help of the refinements, it could be improving gradually and become complicated enough. The term refinement applies to dynamic parts of the model, so-called machines. The most famous example of the Event-B model is the “Controlling Cars on a Bridge” model. It describes the traffic lights for cars crossing the bridge from mainland to island and vice versa. The first model developed for this study case was really simple, it had island and bridge joint together and only two colors in traffic lights: red and green, while the final model had not only traffic lights, but also car censors and all three parts of the study case – island, bridge and mainland. This example shows the idea of the refinement – gradual improvement of the model using the refinements.

**2.1.2 Events**

The main part of machines in Event-B is event. At the beginning of development there is just one event in each created machine – INITIALISATION event. This event is used for initializing actions (variables, invariants (?)). No one model can work properly without this event. While developing the final model, different events can be created to describe the model. Each event should describe the one action in real life. In given example “Controlling Cars on a Bridge” there are different events describing “A car is leaving the mainland and entering the Island-Bridge”, “A car leaving the Island-Bridge and re-entering the mainland”, etc. actions. The more precise model becomes the more events it usually includes. Events can have no guards, they can be also simple and guarded (keyword where) or parameterized and guarded (keywords any and where) [2].

**2.2 Theory of institutions and SBOs**

Despite the fact that Event-B is quite popular language and is used in industry, it has a great disadvantage - it doesn’t have well-developed modularization constructs and it is not easy to combine specifications in Event-B with those written in other formalisms [Marie’s paper]. Modularization constructs are the base of the general theory of institutions. What is institution? The concept of institution is introduced to formalize the informal notion of “logical system”. Institutions enable abstracting away from syntactic and semantic detail when working on language structure “in-the-large”; for example, the language features can be defined for building large structures from smaller ones, possibly involving parameters, without commitment to any particular logical system. This applies to both specification languages and programming languages. Institutions also have applications to such areas as database theory and the semantics of artificial and natural languages [1]. A specification is the main modelling unit in an institution, but specification language is not a programming language, this is a collection of sentences about programs [1]. For a (pure) logical programming language, the specification is also a program [1].

The key concepts in the theory of institutions are:

* A specification is the main modelling unit in an institution.

In terms of Event-B, a specification is referred to as a component: i.e. it is the description for either a machine or a context. In the theory of institutions, a specification consists of a signature along with a set of sentences over that signature.

* The signature of a specification is the set of names used in that specification.

For an Event-B machine this is the set of (global) variables and event names. For an Event-B context this is the set of constant and set names. We don't worry about the names of invariants, guards and axioms, since these are just labels for information and can't be seen by other specification.

* The sentences in a specification are just the predicates that define things.

In Event-B machines these are the invariants, guards and actions; in Event-B contexts they are the axioms.

**2.3 Existing Event-B plugins**

Rodin Platform, which supports Event-B language, is simple and easy-to-use tool, but its’ functionality is limited. This limitation doesn’t allow users to manipulate with Event-B elements and use all the features of this language. Developers from all over the world try to make the use of this tool as simple as it is possible. They provide plugins which allow user to only to edit the text of Event-B program (Camille editor), but also to create visual representations of the models (UML-B plugin), animate proofs (ProB animator), rename model elements and make compositions of several models. These plugins allow developers to create very precise and smart models and discover new features of the Event-B language. More details about two plugins close related to the plugin described in this thesis follow:

**2.3.1 Refactoring framework**

The initial work of the Refactoring framework was done by Sonja Holl [3]. The author identified the problems related with the refactoring formal specification languages like B (and Event-B) due to the presence of the proofs. The refactoring of such languages should be very good to avoid breaking proofs. This plugin was created to provide a possibility of renaming elements of the models written in Event-B. Users can rename not only machines and contexts, but also variables, variants, invariants, events, etc. via this plugin. To do so user should right click on the element he wants to rename and choose the new name. This simplicity allows not to worry about the proof obligations crash, because the plugin works very carefully and renames elements in that way so proofs don’t break.

The plugin has several updates, the latest version is 1.3.0 and based on Rodin 3.0.x. During the dozens of updates, the plugin became powerful tool with good functionality and user-friendly interface. It includes not only renaming for main parts of the Event-B model such as machines and contexts, but also small parts of these parts, such as variables, invariants, events, constants, axioms, etc. It also allows users to keep the valid proofs during the renaming. The plugin during the execution operates with three trees: dependency tree to match all the dependencies between the renaming object and other objects in model (such as variables in invariants or theorems), abstract syntax tree of the Event-B language to get access to all of the elements of the model and proof tree to make changes in proof obligations without breaking them. Even though the plugin main functionality is simple renaming, it allows users not to waste their time on creating new elements with other names and deleting unnecessary elements which can cause the crash of the whole system. Figures 2.1-2.4 contains screenshots of the user interface of this plugin showing the main flow of the interaction between user and the system. The figure 2.1 represents the plugin call, caused by right-clicking on the renaming element, figure 2.2 shows the dialog window asking user to input the new name. After the valid input of the new name, the plugin checks if any problems can appear during the renaming (figure 2.3) and start renaming (figure 2.4).

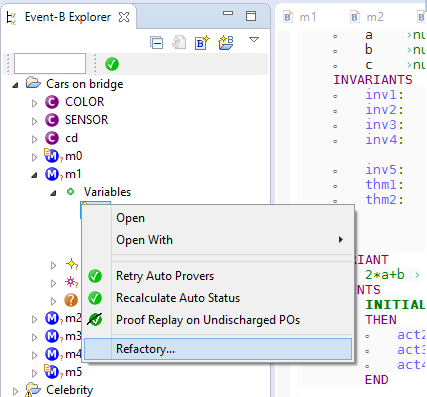


Fig. 2.1 Refactoring (renaming) plugin call

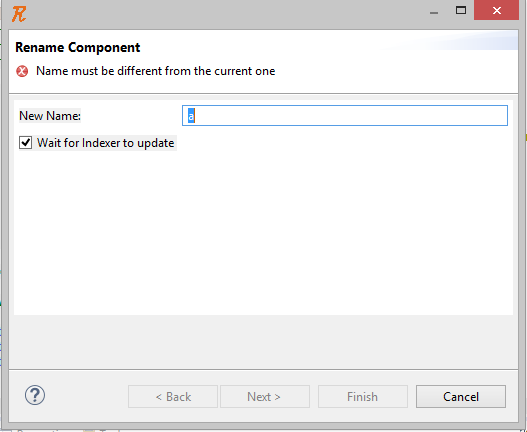


Fig 2.2 New name input

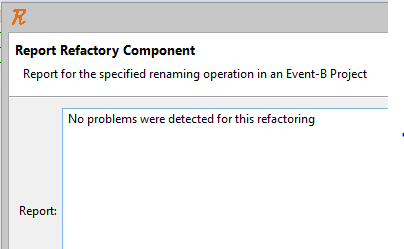


Fig 2.3 Problem report

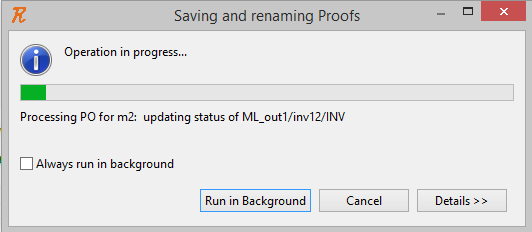


Fig 2.4 Renaming in progress

**2.3.2 Feature composition plugin**

Feature composition plugin was developed by Ali Gondal (University of Southampton) and compatible with Rodin 2.0. This plugin allows user to build a composition model of the input models. The new model is also an Event-B model and is saved in Rodin database. This smart plugin highlights the conflicts between the joining models such as declarations of the same events or variables in both models. It also allows to resolve conflicting situation by removing the repeating/redundant information in different models. The composition editor also provides option for merging events [http://wiki.event-b.org/index.php/Feature\_Composition\_Plug-in].

The composition of several models created by this plugin allows to obtain the model with necessary properties without long manual development. The dialog window of the feature composition plugin shows all available elements such as variables, invariants, events, etc. which can be used in the final composition model. User can select/deselect these elements depending on planned idea of the final model. Besides the possibility of making the composition, plugin allows to merge two or more events by creating a new event. Plugin doesn’t allow to compose variants and theorems. The tool is also capable of composing features at different refinement levels. The composite feature is a typical Event-B model and is automatically checked by the RODIN static checker for any errors [4]. This plugin is a prototype project of the feature composition tool and will be improved by the developers. It was created because the feature composition tools existed before didn’t use all of the capabilities of the Event-B language and provided user with small number of features. New versions of this plugin will be able to deal with proof obligations and create compositions based on existing proof obligations.

**CHAPTER 3**

**Solution**

**3.1 Requirements specification**

The requirements to the generated plugin based on the research questions, introduced in chapter 1 and existing projects, which were mentioned in chapter 2. There are 5 requirements stated below.

**R1 – The plugin should be easy to use for all the users of Rodin Platform**

Despite the fact that all verification languages based on the same principles and use the same logic laws, the syntax of these languages can vary significantly. The user of the plugin should have some idea of working with Event-B and be able to manipulate with its elements. The plugin will help any user to rename any element of the existing model without difficult and time-consuming manipulations. Even if the user hasn’t been working with the Rodin Platform for a long time, he will be able to call the plugin and write commands, based on specification-based operators. If the Rodin Platform user is more advanced, he will be able to learn specification-based operators and combine Event-B specifications with other languages specifications easily.

**R2 – The plugin should allow to rename Event-B elements and compose machines**

The main features of the plugin are renaming of the Event-B elements and composing of machines. As there are existing plugins with the same functionality, but working separately, the goal of this plugin development was to combine features of these two plugins into one plugin and to make unusual way of communication with the system. The implementation of the specification-based operators in Rodin Platform will help to make the Event-B language more ‘standard’ and will allow to combine specifications in Event-B with other formal languages specifications. This standardization will allow developers using different formal languages to work on one project simultaneously.

**R3 – The plugin should not allow to make changes if the input is incorrect**

The main idea of this part is to prevent program execution if the user input is incorrect. As Event-B provides the proofs of the system, incorrect data input can cause breaks of these proofs. The plugin should not only be user-friendly and simplify models development, but it also should keep the system stable and not allow to execute operations which can cause the break of the proof obligations.

**R4 – The plugin should give user information about incorrect data input**

This part is very important in software development. The feedback from the program shows the user any possible mistake he could make. There are several common mistakes that could be identified by the plugin. First of all, user can write element name incorrectly and the plugin display the message that the element with the given name doesn’t exist. Another way of making a mistake is to specify the same name for the renaming element as it already has or do not specify new name at all. Third possible mistake is misspelling of the key words or placing them in the wrong place of the command. As the Event-B language is case-sensitive, all the commands should be written in lower-case and the names of renaming elements should match the elements name in the model. In other case the plugin will not identify elements.

**R5 – The plugin should manipulate with Event-B models by specification-based operators**

As was mentioned above, the main disadvantage of the Event-B language is that it doesn’t have well-developed modularization constructs and it is not easy to combine specifications in Event-B with those written in other formal languages. The main feature of this plugin is the support of the specification-based operators. This feature allows developers not to worry about the Event-B language structure. Plugin executes all the operations using the Event-B abstract syntax tree, so all the dependencies between elements are considered and in case of valid input, no one machine will break during the renaming. Within the project, not all of the specification-based operators were implemented in the plugin, only part of them, but even this part could show the advantages of the chosen approach. After the renaming, elements in the tree will change their names automatically, without the necessity of refreshing the model tree. The one thing user will need to do is to open the renamed element (or machine/context if the element is inside them) and save it manually. After the saving, some errors can occur in the model tree, but after the full build of the workspace, all errors will disappear. If user changes the name of the element which is used somewhere in the model (e.g. variable can be used in invariants or events), after the renaming it will be changed in all occurrences. This is the main advantage of the renaming with the help of this plugin. During the manual renaming of the elements, user can forget to change the name somewhere and this will cause errors. Manual renaming is very time-consuming as well.

**References**

[1] J. A. Goguen and R. M. Burstall. Institutions: Abstract Model Theory for Specification and Programming. Journal of the A.C.M., 39(1):95{146, January 1992

[2] J.-R. Abrial. The Event-B Modelling Notation, October 2007

[3] S.Holl. Refactoring of B models. Bachelor thesis, 2007, pp.1-52

[4] A. Gondal, M. Poppleton, C. Snook. Feature Composition – Towards product lines of Event-B models