Abstract - The software requirements engineering (RE) process is one of the key processes in software development. The aim of requirements engineering process is capturing, understanding and analyzing customer requirements. Today, use cases are widely used as a technique for specification of the functional requirements of the system. Even through use cases are widely used for specification of the functional requirements, their semantics are unclear that it is difficult to apply them to complex problems. In this paper we have proposed SilabReq language for use cases specification for a standard information system. It is developed under XText framework. The benefits of using SilabReq for use cases specification are reflected in the fact that software requirements are considered as a model. Thereafter, the use cases can be used for automatic processing and analyzing.

Keywords-requirements engineering; software requirements; use case;use case specification; (key words)

I. INTRODUCTION

The software requirements engineering (RE) process is one of the key processes in software development. An important aim of requirements engineering is capturing, understanding and analyzing customer requirements.

Different authors define requirements differently. Requirements are defined as a property that must be exhibited in order to solve some real-world problem [1] or needs and constraints placed on a software product that contribute to the solution of some real-world problem [2]. Another definition of a requirement is given in IEEE Std 610.12-1990 [3]: (1) A condition or capability needed by a user to solve a problem or achieve an objective. (2) A condition or capability that must be met or processed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents.

The requirements can be presented in different ways. Herman [4] emphasizes two forms of requirements representation, as follows: descriptive requirements presentation and model-based presentation of the requirements. He makes the distinction between Natural Language Requirement Representation and Constrained Language Requirement Representation as specializations of Descriptive Requirement Representation; as well as UML-Based Requirement Representation as a specialization of Model-Based Requirement Representation.

One of the problems in RE is that these customer requirements are not traced to software specification. Therefore, a common and still open problem in RE is how to derive software specification from requirements. In order to solve this problem we need to represent requirements in a form understandable for all participants in the software project and which will provide certain level of automation in requirements handling. Furthermore, the integration of use cases within Model-Driven Software Engineering requires a better definition of the use case contents, particularly description of sequences of action steps, pre- and post-conditions, and relationship between use case models and conceptual models.[5]

In order to achieve this goal we have developed a language for presentation software requirements in the use case form. This language (called as SilabReq) is expressed in a formal way through a XText grammar. [6] XText grammar can be expressed as a meta-model which allow us to treat use cases and their parts as a model. This model can be used for automatic transformations.

The paper is organized as follows. In the next section, we put our work in the context of some related works. Section 3 gives an overview into use cases and use case model and how it relates with our work. In Section 4, we present our language for functional software requirements presentation in use case form with some examples. Finally, Section 5 concludes the paper and outlines some future works.

II. RELATED WORKS

Originally, use cases were developed as a technique for capturing the required behavior of a software system [7]. Today, it is widely used as a technique for functional
requirements specification of the system. One of the key problems with use cases is their semantics, especially precise notation for use cases actions. This paper is primarily focused on this notation.

Jacobson has described steps in a use case as representing a transaction. He has identified four types of transactions:

- The primary actor sends request and data to the system
- The system validates the request and the data
- The system alters its internal state
- The system replies to the actor with the result.

Even through Jacobson identified different actions in use case the specification of these actions are left undefined in UML. In [8], Williams et al. use 4 basic actions (Input, Output, Computation, and Exception Handling) and 4 flow-of-control actions (Selection, Iteration, Inclusion and Extension) in use case specification. Furthermore, they explained common problems that exist in the practice of the use case modeling, as well as the lack of use case semantics, which allows them to be executable and analyzable. They proposed conceptual framework for developing use cases to be useful for industry but they did not propose precise notation for use cases. We follow their approach toward precise use cases and we propose notation for it.

Further, in [9] Genilloud et al. they are explain the limitations of the graphical notation for use cases specification and emphasized that use cases specification in natural language should describe the actors’ actions of entering data, selecting an option, etc., so that they can be easily read by system users and other stakeholders. Apart from explanation of the use cases actions, they have not suggested how to specify these actions.

According to [7, 8, 9] we have identified several types of actions that appear in use case scenarios and divided them into two categories: the actions performed by the user and actions performed by the system. In order to better specify use case, we have developed SilabReq language. This language uses concrete textual syntax, which is defined using an appropriate grammar. The aim of this paper is to present this language, especially grammar for actions in use cases scenario.

In [10], Some defines the abstract syntax for a use case textual presentation. The author emphasizes that certain elements (formalisms) of the UML languages such as actions, activities are formally defined through the meta-model (or the corresponding meta-class) while meta-model for a use cases textual description has not been defined even through text-based notation is a primary notation for use case description. Therefore, the author defines the meta-model to describe the interaction between the user and the system. His work was inspired by a variety of guidelines for writing use cases and various defined forms (template). Our work differs from [10] because he defines use cases meta-model as an extension to the UML specification while we propose a language for a use case specification in general as well as specification for each type of use case actions. Other than this difference, there is a significant difference in a way that how we can generate other artifacts based on the use cases specification. The domain model can be derived in [10] but requires human intervention (semi-automatically), while in our case it can be derived automatically.

Many stakeholders participate in software development process. They play different roles in software development process. In [11] the author describes four different use cases points of view, according to the stakeholder’s role in software development process. Our language is designed to enable use cases specification according to the stakeholder’s roles and according to the user’s specific knowledge (domain knowledge, analyst knowledge etc). This means that use cases specification in SilabReq language is extensible; therefore, stakeholders with different role and knowledge can enrich it. We achieved it, introducing mechanism that enabled that each action in use cases scenario can be presented according to stakeholder’s knowledge. This mechanism is based on the possibility that some parts of use cases can be activated according to stakeholder’s role. Therefore, different stakeholders can work on the same use cases specification and semantically enrich it. In addition, different stakeholders can use different views on a use cases specification because it is possible to create transformation from one use cases point of view to another. For example, the users can use textual presentation of the use cases, the developers can use UML sequence diagram to see actions between the actors and the system or UML class diagram to see domain model.

III. USE CASE AND USE CASE MODEL

Use cases are technique for capturing the functional requirements of a software system. There have been many different definitions of use cases in the literature. Here are some of them:

“They are stories or cases of using a system. Use cases are not exactly requirements or functional specifications, but they illustrate and imply requirements in the stories they tell.” [12]

“A use case is a description of a set of sequences of actions, including variants, that a system performs to yield an observable result of value to an actor.” [13]

Use cases are “narrative” approach to requirements engineering because they describe the context and requirements in natural language. Therefore, use cases is primary a textual representation technique. Each use case consists of one or more scenarios that describe how the system should interact with the user or other systems to achieve a particular goal. UML defines a use case as the specification of a set of actions performed by a system, which yields an observable result that is, typically, of value for one or more actors or other stakeholders of the system [14]. UML uses different types of diagram (sequence diagram, activity, state machine) to specify these actions.

Use cases describe interaction between the users and the systems from the user’s point of view [11]. Each use cases contains one main scenario and zero or more alternative scenarios. Each scenario contains actions which describe
interaction between the users and the system. We have identified these actions and formally described them in SilabReq language. From the user’s point of view three types of these actions are important. The user executes two of them, while system executes the last one. We have named these actions as following:

- **Action Actor (user) Prepares Data for System Operation execution** (first type of user’s action - APDSO)
- **Action Actor Call system to execute System Operation** (second type of user’s action ACSO)
- **Action System Returns Results of system operation execution** (system’s type of action - SRR)

In the next section, we describe these actions in form of BNF grammar.

The use case model is a model of interaction between the users and the system. The use case model contains use cases, actors and the relationships between them. In system requirements modeling, it is commonly recommended that the use case model and the domain model should be used together [15]. In order to satisfy this recommendation, we introduce one element more in the use case model than in UML use case model. We have named it as usecase_domainconcept (Listing 1). This element presents a concept (base domain object) over which the use case is executed. For example, in the Register new student use cases, Student is that concept over which use cases is executed, while in the Register exam results use case, it is ExamResult concept. Each use case is related with only one primary (base domain concept) and zero or more other domain concepts. Therefore, our SilabReq use case model extends UML use case model with this element. In the next section, we present our SilabReq language.

### IV. SILAB PROJECT

Silab Project was initiated in Software Engineering Laboratory at Faculty of Organizational Sciences, University of Belgrade in 2007. The main goal of this project was to enable automated analysis and processing of software requirements in order to achieve automatic generation of different parts of a software system.

At first, the project has been divided in two main subprojects SilabReq and SilabUI projects that were being developing separately. SilabReq project considered formalization of user requirements and transformations to different UML models in order to facilitate the analyses process and to assure the validity and consistency of software requirements. SilabReq language is the main part of this project. On the other hand, SilabUI project considered impacts of particular elements of software requirements and data models on resulting user interface in order to develop a software tool that enables automatic generation of user interface based on the use case specification and the domain model.

When both subprojects reached desired level of maturity, they were integrated in a way that some results of SilabReq project can be used as input for SilabUI project. As a proof of the concept, Silab project has been used for the Kostmod 4.0 project, which was implemented for the needs of the Royal Norwegian Ministry of Defense.

The SilabReq project includes SilabReq Language, SilabReq Transformation and SilabReq Visualization components. These components are shown in Fig. 1.

![Figure 1. Main components of the SilabReq project](image)

The SilabReq Language component presents our SilabReq controlled natural language for use case specification. In the following section, we have paid special attention to SilabReq language.

The SilabReq Transformation component is responsible for transformation software requirements into different models. Recently, we have developed transformations that transform SilabReq model into an appropriate UML model. These transformations are:

- **SilabReqUCConceptModel transformation** generates domain model (T1)
- **SilabReqUCSystemOperation transformation** generates system operations
- **SilabReqUC transformation generates UML use case model (T3)**
- **SilabReqUC Sequence transformation** generates UML sequence model
- **SilabReqUC StateMachine transformation** generates UML state-machine model (T2)
- **SilabReqUC Activity transformation** generates UML activity model (T4)

The figure below (Fig. 2) presents some of these transformations.
All of these transformations are defined through Kermeta language for meta-modeling. Kermeta is the model-oriented language. Kermeta meta-model is fully compatible with Essential OMG Meta-Object Facility (EMOF) meta-model and Ecore meta-model, which is part of the Eclipse Modeling Framework (Eclipse Modeling Framework EMF). Fig. 3 shows some of these transformations.

The SilabReq Visualisation component is responsible for visual presentation of the specified software requirements. Recently, we have developed only UML presentation of these requirements. So, we can present SilabReq use cases through UML use case, UML sequence, UML activity or UML state-machine diagram.

V. SilabReq LANGUAGE FOR USE CASE SPECIFICATION

Use cases are text documents, not diagrams, and use-case modeling is primarily an act of writing text, not drawing diagrams [16]. In order to satisfy this approach we have developed SilabReq language for precise use case specification. It is developed under XText framework [17]. This framework is based on openArchitectureWare generator [18] framework, the Eclipse Modeling Framework (Eclipse Modeling Framework) [19] and Another Tool for Language Recognition (ANTLR) parser generator [20]. The framework uses the BNF grammar for the description of concrete syntax of the language. On the other hand, based on the BNF grammar, the framework creates a meta-model that describes the abstract syntax of the language. Language definition starts with the definition of the context free grammar of the language.

Kleppe defines the language as "a language description of language L is the set of rules according to which the linguistic utterances of L are structured, optionally combined with a description of the intended meaning of the linguistic utterances" [21]. According to the Kleppe’s definition, defining a language involves the definition of the rules that are used to create a different structure of the language (language syntax), while defining the meaning of these terms (semantics of the language) is optional. Defining the syntax of the language involves defining concrete and abstract syntax of the language. The abstract syntax of the language describes the concepts that appear in the language, their relationships, regardless of their presentation. On the other hand, the concrete syntax provides a presentation of the language concepts defined by using abstract syntax.

Different approaches for the language definition use different formalisms to define abstract and concrete syntax of the language. Xtext framework uses the XText grammar for description of the concrete syntax of the language. On the other hand, based on the BNF grammar, the framework creates a meta-model that describes the abstract syntax of the language.

SilabReq grammar starts with defining SilabReqUseCaseModel a root parser rule. It defines actors (through Actor parser rule), use cases (through UseCase parser rule) as well as discovered domain concepts (through DomainConcept parser rule). Most of these domain concepts will appear in the final domain model. The listing below describes grammar of the SilabReqUseCaseModel and related parser rules.

```
Listing 1. The grammar of the SilabReqUseCaseModel and related parser rules

UseCase parser rule starts with defining a use case with a unique identifier and a use case name. Each use case is related to one base domain concept with established cross reference (named usecase_domainconcept), a precondition (which is optional) and use case execution flow.

UseCaseFlow parser rule defines execution of the use case. Each use case flow contains of one or more use case action block (defined through UseCaseActionBlock parser rule). The use case action block contains:

- Actions which performed by the users (defined through UserActionBlock parser rule)
- Actions which performed by the system (defined through SystemActionBlock parser rule).
```

The figure below (Fig. 3) presents a meta-model of the UseCaseActionBlock parser rule.
Therefore, we follow use case flow execution as execution of the block actions, that are performed by the users and the system. These actions are grouped into the use case action blocks. One use case flow contains one user and one system action block. The user action block can contain one or more actions \textit{Actor Prepares Data for System Operation execution} and only one action \textit{Actor Call system to execute System Operation}. The system action block contains only one action \textit{System Returns Results of system operation execution}.

The user action block is defined through \texttt{UserActionBlock} parser rule. It contains zero or more user action flow \texttt{(UserActionFlow)}. Therefore, we follow the execution of the user actions through \texttt{UserActionFlow} parser rule. The grammar of this parser rules is given in the listing below (Listing 2).

\begin{verbatim}
UserActionBlock:
  "USER ACTIONS:" (user_actionflow+=UserActionFlow)*;
\end{verbatim}

Listing 2. The grammar of the UserActionBlock parser rule

Each of the user actions in the user action flow can be executed sequentially, iteratively or conditionally. In order to enable this approach, we have developed \texttt{SequenceUserAction}, \texttt{IterateUserAction} and \texttt{ChooseUserAction} parser rules. The grammar of these parser rules are given in the listing below.

\begin{verbatim}
UserActionBlock:
  "USER ACTIONS:" (user_actionflow+=UserActionFlow)*;
UserActionFlow:
  SequenceUserAction|IterateUserAction|ChooseUserAction;
SequenceUserAction:
  sequence_useraction = UserAction;
IterateUserAction:
  "iterate" (user_action += UserActionFlow)* "end iterate";
ChooseUserAction:
  "choose" if' condition=STRING "then" {userAction += UserActionFlow}+ "end choose";
\end{verbatim}

Listing 3. The grammar of the UserActionFlow and related parser rules

The figure bellow (Fig. 4) presents the meta-model of \texttt{UserActionFlow} parser rule.

\begin{verbatim}
UserAction:
  UserActionType|DirectiveStep;
UserActionType:
  ActorPrepareDataSOAction|ActorCallSOAction;
ActorPrepareDataSOAction:
  'apdso: ' "stepNumber=INT" actor_in_action=[Actor] user_enterdata=UserEnterData;
  (T_UPPER_WORD (user_enters+=UserEnters)* )*;
UserEnterData:
  DomainConceptEnters | FreeEnters;
DomainConceptEnters:
  entity_name=[DomainConcept] entity_property=STRING;
FreeEnters:
  enters=T_LOWER_WORD;
ActorCallSOAction:
  'acso: ' "numStep=INT" "actor_in_action=[Actor] calls system to to" actor_call_so = ActorCallSO;
  (T_LOWER_WORD (systemoperation+=SystemOperation)*)*;
SystemOperation:
  NewOperation|ExistOperation;
\end{verbatim}

Listing 4. The grammar of the UserAction parser rule

The user actions are defined using \texttt{UserAction} parser rule. As we have emphasized in the previous section, we have identified two types of the user actions: \textit{Actor Prepares Data for System Operation execution} and \textit{Actor Call system to execute System Operation}. Both of these actions are defined through appropriate parser rules \texttt{ActorPrepareDataSOAction} and \texttt{ActorCallSOAction} respectively. The grammar of the \texttt{UserAction} rule is presented in the listing bellow.

\begin{verbatim}
UserAction:
  UserActionType|DirectiveStep;
UserActionType:
  ActorPrepareDataSOAction|ActorCallSOAction;
ActorPrepareDataSOAction:
  'apdso: ' "stepNumber=INT" actor_in_action=[Actor] user_enterdata=UserEnterData;
  (T_UPPER_WORD (user_enters+=UserEnters)* )*;
UserEnterData:
  DomainConceptEnters | FreeEnters;
DomainConceptEnters:
  entity_name=[DomainConcept] entity_property=STRING;
FreeEnters:
  enters=T_LOWER_WORD;
ActorCallSOAction:
  'acso: ' "numStep=INT" "actor_in_action=[Actor] calls system to to" actor_call_so = ActorCallSO;
  (T_LOWER_WORD (systemoperation+=SystemOperation)*)*;
SystemOperation:
  NewOperation|ExistOperation;
\end{verbatim}

Listing 5. The grammar of the UserAction parser rule
Listing 4. The grammar of the UserAction and related parser rules

Definition of the action Actor Prepares Data for System Operation execution starts with specifying the actor and the data entered by actor. The action number is created automatically. The data is defined through UserEnterData rule. This parser rule enables the user to specify the data that enters, either the domain concept with/without domain concept properties (defined through DomainConceptEnters parser rule) or some other data defined through FreeEnters parser rule. The action Actor Call system to execute System Operation is actually the specification of the system operation that user occurs on the system.

In order to enable "include" and "extends" UML use case relationships we have defined DirectiveActionType parser rule. The figure below (Fig.5) describes the meta-model of this rule and related meta-classes.

![Figure 5. The meta-model of the DirectiveActionType parser rule](image)

DirectiveActionType parser rule is an abstract parser rule. It is implemented through ExtensionPoint parser rule that we have used to define extends use case relationship and Include parser rule to define include use case relationship. The grammar of the DirectiveActionType rule is presented in the listing bellow.

```
DirectiveActionType:
ExtensionPoint|Include;
ExtensionPoint:
'extension point:' name=STRING extpoint=[UseCase]
description=STRING?;
Include:
'include:' includeUseCase=[UseCase];
```

Listing 5. The grammar of the DirectiveActionType and related parser rules

After the execution of the system operation the system returns to the user either a successful or error message. The message which will be return to the user depends on the result that system operation returns. The successful execution of the system operation is defined through SuccessfulResponse parser rule. The system returns to the user either String message or some data defined through DataResponse parser rule. The error execution of the system operation is defined through ErrorResponse parser rule. Besides an appropriate message after error execution of the system operation, it is needed to define next action which will occur.

The following part of this section contains the specification of two use cases from the Faculty Student Services System. Different users (students, professors and employees) use this system. Students use it to track their activities on the different subjects. The professors use this system to evaluate students work (for example to register student’s course results). The employees use it to manage the basic data for students, professors and subjects.

The first use case is Register new student. This use case is used by employees to register a new student in the system. The specification for this use case from user’s point of view is given bellow.

```
UC: UC_RegisterNewStudent "Register new student"
Actors: Officer
Use case domain concept: Student
USE CASE FLOW:
```

```
ucase_actionblock+=UseCaseActionBlock)+
"END USE CASE FLOW"
UseCaseActionBlock:
user_actionblock=UserActionBlock
system_actionblock=SystemActionBlock;
SystemActionBlock:
"SYSTEM ACTIONS:" ia=IAResponse
"END RESPONSE"
"END SYSTEM ACTIONS";
IAResponse:
responseSuccessful=SuccessfulSystemResponse
(responseError=ErrorResponse)?;
SuccessfulSystemResponse:
"SUCCESSFUL RESPONSE" {responses += SuccessfulResponse}+
"END SUCCESSFUL RESPONSE ";
SuccessfulResponse:
MessageResponse|DataResponse;
MessageResponse:
'Message: ' message=STRING; DataResponse:
'Data: ' (T_UPPER_WORD dataconcept=[DomainConcept])? *
ErrorResponse:
"EXCEPTIONS" (userViewException+=UserViewException)*
"END EXCEPTIONS";
UserViewException:
message=STRING "action" action=STRING;
```
The use cases flow for the Register new student use case contains a pair of the user and the system block. The user action block involves actions executed by the users (actions from 1 to 4). The actions (from 1 to 3) specify data that a user enters before he calls the system to register a new student (the 4-th action). From the user’s point of view, the system executes only one action (System Returns Results of system operation execution). In this use case, if the system successful saves new user, it returns:

- the following message: Student is saved
- data for the last registered student saved

The next use case (Register course results) is used by the actor with Officer rule. Officer uses this use case to register student’s results. In order to specify student’s result, the actor executes 3-rd and 4-th actions for each student iteratively.

**VI. THE FUTURE OF THE RESEARCH**

SilabReq has in its favor the idea of combining the domain concepts with use case descriptions. In order to improve our approach, we will explore how to relate use cases with ontologies. An ontology can be defined as an attempt to formulate an exhaustive and rigorous conceptual scheme within a given domain, containing all relevant entities and their relationships and rules within that domain, so it can be a good way to avoid inconsistency in software requirements. Therefore, ontology in our approach can be used as a specialized representation vocabulary to a given domain.

Potential further research in this area could relate to generating different types of the automated tests. In that way use case can be better validate by testers.

**VII. SUMMARIZE**

According to [5, 8], the use case semantics are poorly defined, especially a notation for specification use case actions, which makes processing and analyzing difficult. In this paper we have proposed SilabReq language for a use case specification. The benefits of using SilabReq for use case specification are reflected in the fact that software requirements are considered as a model. Therefore, the use cases can be used for automatic analyzing. SilabReq language is extensible which means that all participants in the software project can use it according to their role in specifying software requirements. In this paper we have presented SilabReq language primary from the user’s point of view. The use case specification from user’s point of view requires 3 types of use case actions:

- Action Actor (User) Prepares Data for System Operation execution
- Action Actor Call system to execute System Operation
- Action System Returns Results of system operation execution

In our future work we intend to relate the use case specification with specification of the user interface. We intend to provide an independent realization of each use case action to allow different use case presentations for the same use case. Also, we are developing, a use case tool which will include SilabReq language for the use case specification as well as different transformations which will allow us to generate different artifacts from the use case specification.

**ACKNOWLEDGMENT**

The authors are thankful to MNTRS for financial support grant number 174031.

**REFERENCES**


[18] openArchitecturesWare, http://www.openarchitectureware.org

