C&L: Generating Model Based Test Cases from Natural Language Requirements Descriptions

Edgar Sarmiento, Julio Cesar Sampaio do Prado Leite
Departamento de Informática- PUC-Rio
Rio de Janeiro - Brasil
ecalisaya@inf.puc-rio.br, www.inf.puc-rio.br/~julio

Eduardo Almentero
Departamento de Matemática - DEMAT
Universidade Federal Rural do Rio de Janeiro - UFRRJ
Seropédica - Brasil
almentero@ufrrj.br

Abstract— **Software testing tasks are usually time-consuming, especially if one considers complex projects. Requirements engineering artifacts are a valuable starting point for the development of software products, and most of software requirements specifications are written in natural language. This paper presents a tool that implements an approach for generating test cases based on Natural Language (NL) requirements specifications. The C&L tool translates automatically NL requirements descriptions into behavioral models to support automated testing. Our approach is easy to use and aims to decrease the time and the effort with respect to test case generation. Demonstration of the feasibility of the proposed approach is based on an example of use that describes the operation of the C&L tool.**

Index Terms—**Requirements, scenario, lexicon, testing.**

I. Introduction

Software testing is one of the validation techniques most commonly used. This approach improves the quality of the final product, particularly the functional testing, which allows checking that the software behavior meets its functional requirements. However, these activities are still quite expensive and heavy. The automation of this process is a challenging topic.

According to Meudec [14] in the automation of the testing process, we can identify three important tasks: (1) automation of test cases generation process from requirements descriptions, (2) automation of test cases execution and (3) evaluation of results. This paper focuses particularly on the first task.

The model-based testing (MBT) is an alternative to the automation of these tasks, in which test cases are derived from behavioral models [6] described using formal modeling/specification languages and other notations, such as UML [16].

In most of these approaches for generating test cases for model-based systems, testing practitioners usually decompose the system in different ways or use scenarios, after it models are derived for each scenario. The test cases are derived from these intermediate models.

Among the problems presented by model-based approaches are the difficulty and time required to identify scenarios, the corresponding intermediate models and, consequently, the generated test cases. Usually, the initial descriptions of requirements are appropriate inputs to start the testing process, because these allow reducing its cost and enhancing its effectiveness [5,10,11].

The use of formal specification languages facilitates the process of test automation. However, this practice is expensive and not widely used in industrial practice. On the other hand, natural language is widely used in the description of software requirements, because it provides a communication channel among the stakeholders of the software construction process (users/clients/developers/testers).

Natural language is easily associated with some methods and languages for requirements specification, such as use cases models [3] and scenarios [12]. These languages focus on the situation behavior, thus helping their understanding.

In this context, the identification of scenarios, the models associated with these, and the generation of test cases from natural language requirements descriptions are challenging topics, because the tasks involved in this process are costly especially if we consider complex systems.

This paper presents a tool that implements an approach for generating test cases from natural language requirements descriptions. The C&L (Scenarios & Lexicons) tool [2] automatically transforms descriptions of requirements into UML activity diagrams. This diagram is represented as a directed graph, from which test cases are generated using graph search strategies. Thus, our approach provides a visual representation to help the validation of requirements.

The scenario language [12] is used to describe software requirements; a scenario describes the behavior of the application through episodes, these descriptions do reference relevant words or phrases of the application (Lexicon symbols). Relevant words of the application referenced within the scenario descriptions: (1) make explicit the input data and conditions that exercise different test scenarios from initial requirements descriptions, (2) allow the evolution and derivation of models, (3) this information can also be used to derive and reduce the number of test scenarios.

This paper is organized as follows. Section II describes the languages used in this approach, and it describes the implementation of the C&L, its interface and functionalities. Section III presents the approach implemented in the C&L. Section IV describes an example of use. Section V describes some related work to our proposal, Finally, Section VI presents the conclusions and some suggestions for future work.

II. Background

In this section we will describe the languages proposed in [12] and used in modeling requirements. The C&L tool, which implements the method proposed in this work, is also presented in this section.

A. Language Extended Lexicon (LEL)

LEL is a language designed to help the elicitation and representation of the language used in the application. This model is based on the idea that each application has a specific language. Each symbol in the lexicon is identified by a name or names (synonyms) and two descriptions: Notion (denotation) explains the literal meaning - what the symbol is, Behavioral Response (connotation) describes the effects and consequences when the symbol is used or referenced in the application. Symbols are classified into four types: Subject, Object, Verb and State. Table I shows the properties of a LEL symbol.

In [7] and [18], relevant terms of the application are described only by the name attribute as operational variables and as project glossary terms, respectively.

TABLE I. SYMBOL DEFINITION IN LEXICON LANGUAGE.

Name	Symbol of LEL
Type	Subject/Object/ Verb/State
Synonymous	Term of LEL/Entry/Symbol
Notion	Word or relevant phrase of the Universe of Discourse. It's described by Name, Type, Notion, Synonymous and Behavioral Response.
Behavioral Response	Its description contains the Type. It has zero or more Synonymous.

B. Scenario

Scenario is a language used to help the understanding of the requirements of the application; it is easy to understand by the developers and clients. Scenarios represent a partial description of the application behavior that occurs at a given moment in a specific geographical context - a situation [12,13].

There are different models of scenario [12,13]. In this work, the scenario model is based on a semi-structured natural language [12], and it is composed of the entities described in Table II.

TABLE II. COMPARING SCENARIO AND USE CASE.

Scenario	Description	Use Case
Title	Identifies the scenario. Must be unique.	Use Case #
Goal	Describe the purpose of the scenario.	Goal
Context	Describes the scenario initial state.	Scope
	Must be described through at least one of these options:	Level
	precondition, geographical or temporal location.	Preconditions
Resources	Passive entities used by the scenario to achieve its goal. Resources must appear in at least one of the episodes.	Trigger
Actors	Active entities directly involved with the situation. Actors must appear in at least one of the episodes.	Actors
Episodes	Sequential sentences in chronological order with the participation of actors and use of resources.	Description
Exception	Situations that prevent the proper course of the scenario.	Extensions
	Its treatment should be described.	Sub-Variations
Constraint	Non-functional aspects that qualify/restrict the quality with witch the goal is achieved. These aspects are applied to the context, resources or episodes.	

Use case [3] is a particular model of scenario; however, use case represents specific situations between the user and the system through interface. Scenario describes: situations in the environment and the system; interactions among objects or modules; procedures or methods. Table II explains how a scenario [12] can be also used as a use case [3].

A scenario must satisfy a goal that is reached by performing its episodes. Episodes represent the main course of actions but they also include alternatives. Episodes are: Simple episodes are those necessary to complete the scenario; Conditional episodes are those whose occurrence depends on a specified condition (*IF <Condition> THEN <Episode Sentence>*); Optional episodes are those that may or may not take place depending on conditions that cannot be detailed (*[<Episode Sentence>]*)

A sequence of episodes implies a precedence order, but a non-sequential order can be bounded by the symbol "#", it is used to describe parallel or concurrent episodes *(#<Episode Series>#)*.

While performing episodes, exceptions may arise. They (*Cause[(Solution)]*) are any event arose from an episode and treated by a Solution, it hinders the execution of the episodes. An alternative flow can be represented as a conditional episode (*IF THEN*), or as an exception, where cause is the condition and the solution is described as a simple sentence or in other sub-scenario (alternative flow).

Scenarios are related to other scenarios by integration scenarios or by sub-scenarios, a sub-scenario describes complex episodes, solutions to exceptions, constraints, pre-conditions and alternative flow of actions.

Integration scenarios give a global vision of the application. Integration scenarios give an overview of the relationship among several scenarios, since each integration scenario episode corresponds to a scenario.

Lexicon symbols are referenced into scenario descriptions; underlined UPPERCASE words or phrases are other scenarios and underlined lowercase words or phrases are lexicon symbols.

Table III describes a scenario of the C&L system (integration scenario). Here, a User interacts with the C&L System. The user starts the request by inserting his/her login and password. The C&L System must verify the identification of the user to proceed. If the verification fails the access is denied, and the system shows the "ADD USER" message. Otherwise, the user can perform some operations in the system until the user logout the system. Add collaborator, add lexicon symbols and add scenario operations can be done concurrently.

TABLE III. C&L SYSTEM SCENARIO

Title	C&L System
Goal	Enable the collaboration between different users in the elaboration of projects using scenarios and symbols of the lexicon.
Context	The user needs to elicit the language and the situations of a specific domain.
Resources	List of projects
Actors	User, System
Episodes	1. User LOGIN THE SYSTEM. 2. User SELECT PROJECT or ADD PROJECT. 3. #User ADD COLLABORATOR to the project 4. User ADD LEXICON SYMBOL to the project 5. User ADD SCENARIO to the project.# 6. User LOGOUT THE SYSTEM.
Exception	1.1. If user not in the system then ADD USER. 2.1. If list of projects is empty then the user must ADD PROJECT.

C. C&L (Cenários & Léxicos)

C&L is a Web application developed in the Lua language [22]. The Kepler platform [22] was used to develop C&L because originally Lua was not designed for the development of Web applications. This platform provides a series of modules and tools which facilitates the writing of Lua code for Web.

The C&L architecture [1] is based on the MVC framework. The architecture is vertically divided in layers and horizontally divided in modules. The modules are distributed in the view, controller and model layers, as can be seen in Fig. 1. Four main modules were created from the scenarios that describe the situations of the application: user, project, LEL and scenario. These modules groups functionalities to manage users (user), scenarios (scenario), lexicon symbols (LEL) and projects (project).

The Util layer, proposed in this work, groups the functions responsible to manipulate graphs and strings. This is necessary to enable the model's transformations methods.

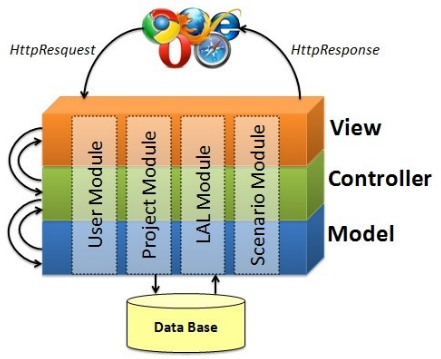
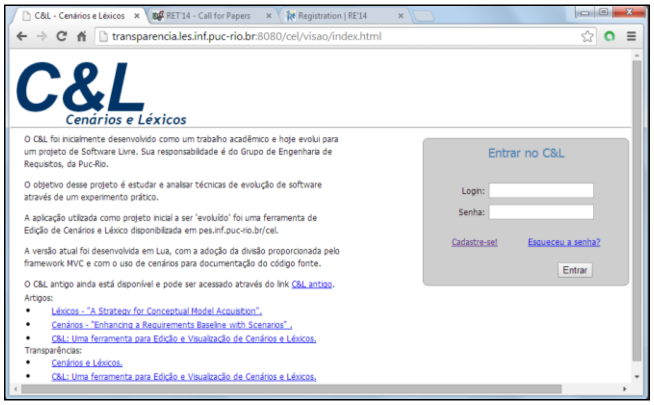


Fig. 1. C&L Architecture.

In the C&L initial page (Fig. 2) the user finds a small text explaining the software and links to external information about the C&L.

To start using the C&L the user must sign up. The user registration is done through a simple form were the user must provide, among other information, its name, e-mail, login and password. After registering in the application, the user can login informing the necessary data.



O C&L foi inicialmente desenvolvido como um trabalho acadêmico e hoje evolui para um projeto de Software Livre. Sua responsabilidade é do Grupo de Engenharia de Requisitos, da Puc-Rio.

O objetivo desse projeto é estudar e analisar técnicas de evolução de software através de um experimento prático.

A aplicação utilizada como projeto inicial a ser 'evoluído' foi uma ferramenta de Edição de Cenários e Léxico disponibilizada em pes.inf.puc-rio.br/cel.

A versão atual foi desenvolvida em Lua, com a adoção da divisão proporcionada pelo framework MVC e com o uso de cenários para documentação do código fonte.

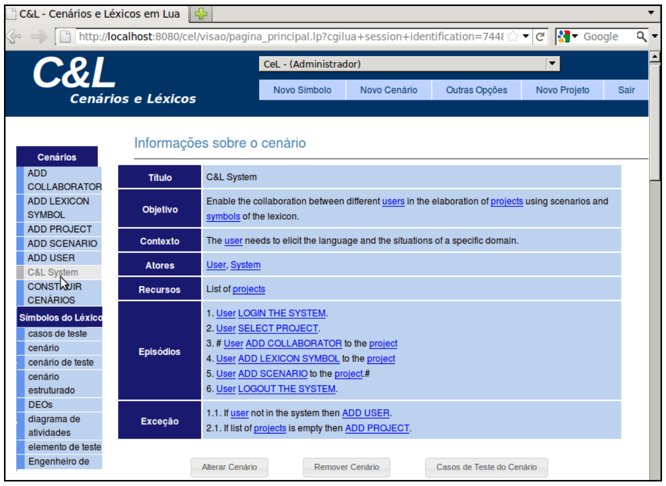
O C&L antigo ainda está disponível e pode ser acessado através do link C&L antigo.

Artigos:
- Léxicos - "A Strategy for Conceptual Model Acquisition".
- Cenários - "Enhancing a Requirements Baseline with Scenarios" .
- C&L: Uma ferramenta para Edição e Visualização de Cenários e Léxicos.

Transparências:
- Cenários e Léxicos.
- C&L: Uma ferramenta para Edição e Visualização de Cenários e Léxicos.

Fig. 2. Initial page of the C&L.

The operation of C&L is described through an integration scenario (Fig. 3 and Table III), which is an artificial scenario created in order to provide an overview of the features of the software. The episodes of this scenario are references to scenarios that describe functionalities provided by the different modules that composes the system. The underlined terms are references to other scenarios (those that appear in uppercase) or to lexicon symbols (those that appear in lowercase). Thus, the term "SELECT PROJECT" in Fig. 3 is a link to the scenario that describes how the user selects a project registered in the system. On the other hand, the term "user" that appears several times in the integration scenario, is a link to the description of the lexicon term whose name is "user". The concept of project is used within the system to represent different domains, where scenarios and lexicon symbols can be grouped.



Informações sobre o cenário

Título	C&L System
Objetivo	Enable the collaboration between different users in the elaboration of projects using scenarios and symbols of the lexicon.
Contexto	The user needs to elicit the language and the situations of a specific domain.
Atores	User, System
Recursos	List of projects
Episódios	1. User LOGIN THE SYSTEM. 2. User SELECT PROJECT. 3. # User ADD COLLABORATOR to the project 4. User ADD LEXICON SYMBOL to the project 5. User ADD SCENARIO to the project.# 6. User LOGOUT THE SYSTEM.
Exceção	1.1. If user not in the system then ADD USER. 2.1. If list of projects is empty then ADD PROJECT.

Fig. 3. Integration scenario and main page of the C&L.

After the user signing in, the system presents its main interface (Fig. 3). In this interface there are many important elements to explore the system functionalities. These elements are: project menu, lexicon and scenarios menu and work area. The first element is located at the top right of the interface. It is through this menu that the user selects the project he wants to work with at the moment.

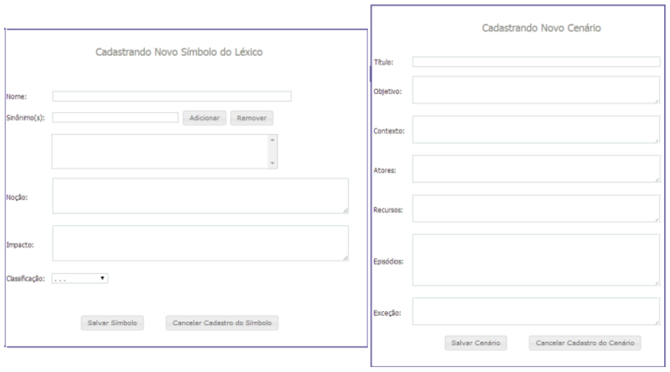


Fig. 4. Add lexicon symbol and add scenario forms.

Three new options appear at the main menu after selecting a project: new lexicon symbol, new scenario and other options. If the user selects the option "new scenario" he is redirected to a form (Fig. 4) to include the information about the new scenario. The same happens when the "new lexicon symbol" option is selected, but in this case the form allows including information about a lexicon symbol (Fig. 4). As the scenarios and lexicon symbols are included in the projects, their title and names are displayed in the lexicons and scenarios menu (left side of the main interface).

When the user selects a lexicon symbol or scenario from lexicons and scenarios menu, the C&L application automatically assembles a network of relationships identifying what scenarios and lexicons symbols are referenced in the body of the selected element. The relationships identified are used to create two kinds of trace: backward and forward. These traces allow the navigation between the elements referenced by the one being visualized (forward) and the navigation between the elements that references the one being visualized (backward). The C&L differentiates the links between scenarios (the scenarios are written in uppercase) and lexicon symbols (written in lowercase). The links between scenarios and scenarios, scenarios and lexicon symbols and lexicon symbols and lexicon symbols are created. Those between from lexicons to scenarios are not created.

The test case generation functionality is activated in the scenario visualization interface (when the user selects a scenario). This functionality generates: a test activity diagram, the test scenarios, test variables and their conditions and a description of the test cases. The transformation methods are implemented as situations of the application in the scenario module. As an example, Figure 7 and Table IV to Table VII show the test artifacts generated for the C&L integration scenario. These test cases exercise the different system functionalities.

III. Test Case Generation Method

This section describes the activities for automation of test case generation from natural language requirements descriptions (Fig. 5). Requirements engineers start it by describing requirements as scenarios and the relevant words or phrases of the application as lexicon symbols [12]. Initially, scenarios are described using natural language; these scenarios are transformed into activity diagrams. Activity diagram paths are generated from interactions among episodes, exceptions and constraints of a scenario. This activity diagram is used for the generation of initial test scenarios using graph-search and path-combination strategies. Scenario descriptions reference lexicon symbols and they represent the input variables, conditions and expected results of test cases. The generation of test values is not covered by this work.

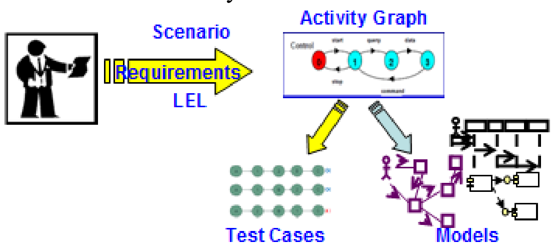


Fig. 5. Test case generation process.

Figure 6 depicts the sequence of tasks implemented in the C&L tool for automation of test cases generation process depicted in Fig. 1. These tasks implement the algorithms to: (1) Transform requirements descriptions into activity diagrams, (2) Generate test scenarios from these activity diagrams, (3) Identify test variables and their conditions from scenarios, and (4) Generate test cases from these activity diagrams and scenarios.

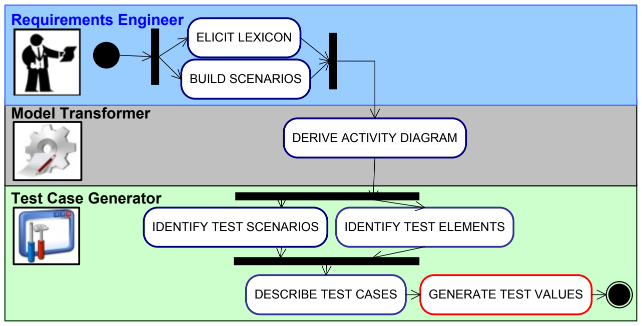


Fig. 6. Automation of test case generation process [17].

Some researches focus on automatic derivation of models from scenario representations, especially in the derivation of state machine models. In [8, 9, 13] LSC (Live Sequence Charts) models as scenario representations are used. These LSC representations are visual behavioral models.

State machine derivation from natural language scenarios facilitates the validation of models because the user/client can monitor the requirements execution [4, 13]; and it also facilitates the derivation of consistent test cases because behavioral models increase the test coverage [18].

The overall procedure for generating test cases is as below:

- ***Step 1:*** Build scenarios and lexicon to represent requirements specifications of the application (manual);
- ***Step 2:*** Derive activity diagrams from given scenarios (automatic):
  - Create the initial node;
  - Derive individual actions nodes from each scenario episodes (*episode sentence*);
  - Derive decisions and control flows from conditional and optional scenario episodes (*condition/option*);
  - Derive individual actions nodes from each *solution* of scenario exceptions;
  - Derive decisions and control flows from each *cause*/condition of scenario exceptions;
  - Derive decisions and control flows from each *constraint* defined in scenario context, resources and episodes;
  - Derive fork-join structures from sequence of episodes bounded by the symbol "#"; link the derived actions as concurrent actions;
  - Create the final nodes;
- ***Step 3:*** Identify test scenarios (automatic):
  - Scan all sequential paths from initial node to final nodes by the DFS (depth-first search) strategy;

- If sequential path contains fork-join structures:
  - Scan all concurrent sub-paths from a fork-join structure by the BFS (breadth-first search) strategy;
  - Combine concurrent sub-paths using sharing resources criterion (in development);
  - Replace combined concurrent sub-paths into sequential path (in development);
- ***Step 4:*** Identify test elements (automatic):
  - Identify test variables from scenario *actors* and *resources*;
  - Identify test variable conditions from scenario *constraints*, *condition/option* of episodes, *cause* of exceptions;
  - Identify initial expected results from scenario goal;
- ***Step 5:*** Describe test cases (automatic):
  - Describe test cases as in [11]; a test case is composed of a test scenario, test variables and their conditions, and an expected result.

The *step 1* is carried out (manually) by the requirements engineers. Activities belonging to steps 2, 3, 4 and 5 are performed automatically by the C&L tool.

The proposed algorithm to generate test scenarios uses BFS (Breath-First-search) and DFS (Depth-First-Search). The BFS is useful for traversing the concurrent activities whereas the DFS is useful for traversing the sequential activities. It generates test scenarios for sequential applications.

The tasks to generate test scenarios for scenario descriptions containing concurrent tasks are in development. It will use domain constraints of the application to reduce the number of test scenarios. The number of test scenarios is a problem because these must test all interactions among concurrent tasks. This algorithm will traverse the concurrent activities by BFS and will generate ordered sequences of interactions among concurrent activities. This criterion will test all relevant interactions among concurrent tasks.

Let AD = $\{A,B,M,F,J,K,T,a_0\}$ be an activity diagram derived from scenario C={Title, Goal, Context, Resources, Actors, Episodes, Exceptions, Constraints}. AD represents the visual behavior of C (AD $\Leftrightarrow$ C). Where $A=\{a_1,a_2,...,a_i\}$ is a finite set of actions; $B=\{b_1,b_2,...,b_u\}$ a set of branches; $M=\{m_1,m_2,...,m_v\}$ a set of merges; $F=\{f_1,f_2,...,f_y\}$ a set of forks; $J=\{j_1,j_2,...,j_x\}$ a set of joins; $K=\{k_1,k_2,...,k_w\}$ a set of final nodes; $T=\{t_1,t_2,...,t_z\}$ a set of transitions which satisfies $\forall t \in T$, $t=<c>e \vee t=e$ where $c \in C$, $e \in E$, $C=\{c1,c2,...,cl\}$ is a set of guard conditions, $E=\{e_1,e_2,...,e_s\}$ a set of edges of AD; and a_0 is the unique initial state of AD.

An activity diagram is a directed graph G=(V,E) where $V=\{A,B,M,F,J,K,a_0\}$ is a union of vertices and E={T} is a set of transitions.

If AD={V,E} is an activity diagram derived from a scenario C, the different paths $p_i \in P$ between the initial state and the final nodes of AD represent the finite set of test scenarios, so, a test scenario (ts) is a sequence of vertices and transitions of AD:

Test scenario = path = $p_i = a_0\ t_0\ a_1\ t_1\ ...\ a_n\ t_n\ k$ where:
$a_i \in V \setminus K \wedge t_i \in E \wedge k \in V \cap K$, i=1,2,…,n.

For instance, the activity diagram derived from the scenario described in the Table III is depicted in Fig. 7.

In this activity diagram, the ADD COLLABORATOR, ADD LEXICON SYMBOL and ADD SCENARIO operations are done concurrently. In this case, these operations are independent; they do not have shared resources.

The test scenarios for exercising the adding of a lexicon symbol and a scenario in a selected project are as below:

TS4 = p_4 = Start -> LOGIN THE SYSTEM -> [NOT (User not in the system)] -> SELECT PROJECT -> [NOT (list of projects is empty)] -> ADD LEXICON SYMBOL -> LOGOUT THE SYSTEM -> Final.

TS5 = p_5 = Start -> LOGIN THE SYSTEM -> [NOT (User not in the system)] -> SELECT PROJECT -> [NOT (list of projects is empty)] -> ADD SCENARIO -> LOGOUT THE SYSTEM -> Final.

IV. Example of Use

This section presents the application of our approach/tool using as example the generation of test cases of the main scenario of the C&L tool. It describes the integration scenario of the C&L system; this scenario represents the main sequences of operations that a user can execute in the C&L system: Add collaborator, add lexicon symbol and add scenario operations are done in an indistinct order or concurrently. In this case, these operations are independent; they do not have shared resources.

Figure 7 depicts the behavioral model derived of the integration scenario of the C&L system (Table III). This activity diagram shows the main operations requested by the user of the system: add collaborator, add lexicon symbol and add scenario. It also shows the alternative or exceptional operations executed by the user: add user and add project.

The actions in the activity diagram are labeled using the id of the episode or the exception in which are described. Table IV shows the id and the name of the actions of the activity diagram of Fig. 7.

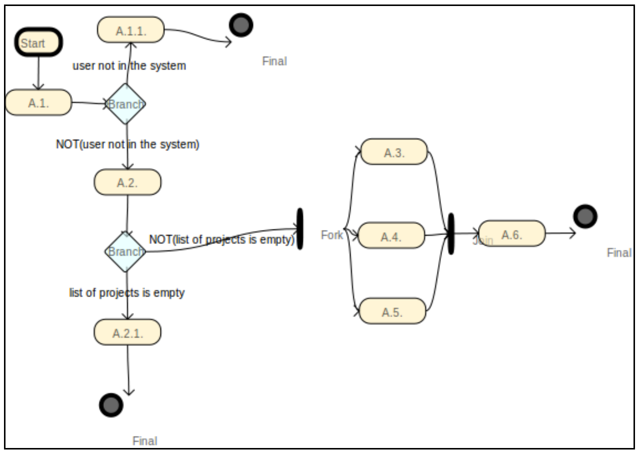


Fig. 7. Activity diagram of the C&L System.

TABLE IV. TEST ACTIONS.

Id	Action Name
A.1.	User LOGIN THE SYSTEM.
A.1.1.	ADD USER.
A.2.	User SELECT PROJECT or ADD PROJECT.
A.2.1.	User ADD PROJECT.
A.3.	User ADD COLLABORATOR to the project
A.4.	User ADD LEXICON SYMBOL to the project
A.5.	User ADD SCENÁRIO to the project
A.6.	User LOGOUT THE SYSTEM.

The input variables and conditions that exercise the test scenarios are extracted from scenario described in Table III. The input variables (IT) are extracted from resources (e.g. List of projects) and from actors (e.g. User and System). Table V shows the variables and their conditions (CD) extracted from the exceptions. When a variable does not have conditions defined in the scenario, or its conditions are not referenced, the C&L tool generates two conditions (*Valid and Not valid*) and consider for the test cases the "*Valid"* condition.

For instance, the conditions of the "*List of projects*" variable are:

List of projects = {list of projects is not empty, Not (list of projects is not empty)}.

TABLE V. TEST VARIABLES AND CONDITIONS.

Id	Variable	Conditions
V1	User	User not in the system
		NOT(User not in the system)
V2	System	Valid
		Not valid
V3	List of projects	list of projects is empty
		NOT(list of projects is empty)

The test scenarios for exercising the main operations on the C&L system are shown in the Table VI. It shows three main execution paths and two alternative/exceptional paths. The alternative/exceptional paths are exercised when the "*user is not authenticated in the system"* or the "*list of the projects is empty"*.

TABLE VI. TEST SCENARIOS

Id	Test scenario
TS1	Start-> A.1. -> [User not in the system]-> A.1.1. -> Final
TS2	Start-> A.1. -> [NOT (User not in the system)]-> A.2. -> [list of projects is empty] -> A.2.1. -> Final
TS3	Start-> A.1. -> [NOT (User not in the system)]-> A.2. -> [NOT (list of projects is empty)]-> A.3. -> A.6. -> Final
TS4	Start-> A.1. -> [NOT (User not in the system)]-> A.2. -> [NOT (list of projects is empty)]-> A.4. -> A.6. -> Final
TS5	Start-> A.1. -> [NOT (User not in the system)]-> A.2. -> [NOT (list of projects is empty)]-> A.5. -> A.6. -> Final

Table VII shows the test cases generated for the "C&L System" scenario. A test case is composed of a test scenario, input variables and their required conditions and an expected result.

The initial set of expected results (ER) for the main flow and the alternative/exceptional flows were extracted from the "Goal": *OK C&L System* and *NOT C&L System*.

From input variables and conditions, we can generate representative values for testing. This process will require human intervention and our approach leaves this open.

TABLE VII. TEST CASES

Id	TS Id	Variables/Conditions			Expected result
		V1	V2	V3	
TC1	TS1	User not in the system	Valid	N/A	Not C&L
TC2	TS2	NOT(User not in the system)	Valid	list of projects is empty	Not C&L
TC3	TS3	NOT(User not in the system)	Valid	NOT(list of projects is empty)	OK C&L
TC4	TS4	NOT(User not in the system)	Valid	NOT(list of projects is empty)	OK C&L
TC5	TS5	NOT(User not in the system)	Valid	NOT(list of projects is empty)	OK C&L

V. RELATED WORK

UML use cases are widely used to specify requirements, however test cases generated from these models are usually described at high level of abstraction, and commonly it is necessary to refine them because external inputs (conditions required to execute test scenarios) are not explicit in the initial descriptions of requirements artifacts used as input. Denger and Medina [5] discuss some approaches for generating test cases from use cases described in natural language; these approaches are incomplete and not automated.

Some approaches have shown how testing tasks can be automated. These approaches generate test cases from activity [7,18] and sequence diagrams [15] derived from use cases descriptions. The test variables referenced in the use cases are described as operational variables [7] and glossary terms [18].

In [15] is proposed an approach to test generation from use case extended with contracts and sequence diagrams; it is necessary create different sequence diagrams to represent use case scenarios. A dedicated OCL (Object Constraint language) is used to describe contracts. In this approach the input artifacts are enriched before automate testing tasks.

Several works have been done for test case generation from UML activity, state and sequence diagrams; however, in most of reviewed works, it is necessary to refine the input models into intermediate models (not automated) to make explicit test inputs or conditions of them.

Some test generation methods based on path analysis of activity diagrams (and variations of it) which contain fork-join structures were proposed.

In [19], a sequence diagram is converted into a concurrent composite graph (variant of an activity diagram); then they applied DFS search technique to traverse the graph, BFS search algorithm is used between fork and join construct to explore all concurrent nodes. The test information is obtained from OCL (Object Constraint language) descriptions. In [20] an activity diagram is mapped to an Input/Output explicit Activity Diagram (explicitly shows external inputs and outputs); this diagram is converted to a directed graph for extraction of test scenarios and test cases (Basic path).

In most of the related works, the mapping task into other models and test information enrichment task are not automated, so they could be expensive and time consuming;

Our approach deals directly with the requirements artifacts produced during the Requirements Engineering (RE) phase, the lexicon and scenarios. No modification is required, as in related works, artifacts resulting from the RE has to be enriched (identification test variables) before using the method. Our approach also deals concurrent applications, by using a scenario language operator for concurrence.

VI. Conclusion

Natural language based requirements specification, like the scenario and LEL techniques explored in this work, helps developers to identify the test cases to exercise the different execution flows of the software. As such, it improves the quality of the product from the initial stages of software production, contributing to the reduction of failures and the reduction of maintenance costs after the final product was delivered.

The functionality of generating test cases from scenarios was developed in order to reduce the costs of the testing process, using a graphical display to support the test scenarios, test elements and test cases.

The use of scenario language [12] to describe software requirements facilitates unit and integration testing because it describes interactions among objects or modules as situations of the system. Modules are described as scenarios of a macro-scenario (System), and modules can be described by other *sub-scenarios* that describe specific situations (procedures). Unit testing is applied on scenarios that describe specific procedures of a module and; integration testing is applied on scenarios that describe modules and their procedures, and system and its modules.

The C&L has been used and evolved by the PUC - Rio requirements engineering group [21]. Methods for model transformation are being improved. Their results are positive and therefore its evolution continues.

In the future, we plan to deal with the testing of exceptions and non-functional requirements; in this work we have drafted some criteria for mapping exceptions and non-functional requirements described (constraints/conditions on resources) in scenarios to behavioral models and testing.

Other future works include: (1) the derived activity diagrams will be represented as XML-XMI document in order to be editable by UML editors, (2) the extraction of input variables and conditions for testing will be improved using natural language processing strategies, and (3) approaches for the verification of scenarios and lexicons according to their specific construction rules will be implemented.

References

[1] E. Almentero, J.C.S.P. Leite, C. Lucena, Towards Software Modularization from Requirements, in ACM SAC, 2014.

[2] C&L, Scenarios & Lexicons, Available at: http://pes.inf.puc-rio.br/cel.

[3] A. Cockburn, Writing Effective Use Cases, Addison-Wesley, Reading, MA, 2001.

[4] C. Damas, B. Lambeau, P. Dupont, A.v. Lamsweerde, Generating annotated behavior models from end-user scenarios, IEEE TSE, volume 31, number 12, 2005.

[5] C. Denger, M. Medina, Test Case Derived from Requirement Specifications, Fraunhofer IESE Report, 2003.

[6] I. K. El-Far, J. A. Whittaker, Model-Based Software Testing, in Marciniak, J. J. (Ed.). Encyclopedia Of Software Engineering. Wiley, 2001.

[7] J. J. Gutiérez, M. J. Escalona, M. Mejías, and J. Torres, An approach to generate test cases from use cases, in Proceedings of the 6th International Conference on Web Engineering, pp. 113-114, 2006.

[8] D. Harel, H. Kugler, A. Pnueli, Synthesis Revisited: Generating Statechart Models from Scenario-Based Requirements, in Formal Methods in Software and Systems Modeling, Lecture Notes in Computer Science, Springer Berlin/Heidelberg, 2005.

[9] S. Uchitel, G. Brunet, M. Chechik, Behaviour Model Synthesis from Properties and Scenarios, in Proceedings of ICSE '07, IEEE Computer Society, pp. 34-43, 2007.

[10] C. Heitmeyer, Formal methods for specifying, validating, and verifying requirements. J Univ Comput Sci, vol. 13, num. 5, pp. 607-618, 2007.

[11] J. Heumann, Generating test cases from use cases, IBM, 2001.

[12] J. C. S. P. Leite, G. Hadad, J. Doorn, G. Kaplan, A scenario construction process, Requirements Engineering Journal, Springer-Verlag London, vol. 5, num. 1, pp. 38-61, 2000.

[13] E. Letier, J. Kramer, J. Magee, and S. Uchitel, Monitoring and Control in Scenario-Based Requirements Analysis, in Proceedings of ICSE, 2005.

[14] C. Meudec, ATGen: automatic test data generation using constraint logic programming and symbolic execution, in Proceedings of Softw. Test., Verif. Reliab. 2001.

[15] C. Nebut, F. Fleurey, Y. Le Traon, J. M. Jezequel, Automatic test generation: A use case driven approach, IEEE Transactions on Software Engineering, vol. 32, num. 3, 2006.

[16] OMG, OMG Unified Modeling Language (OMG UML), Superstructure, V2.1.2. The Object Management Group (OMG), Needham, MA, USA, 2007.

[17] E. Sarmiento, J. C. S. P. Leite, N. Rodriguez, A. von Staa, An Automated Approach of Test Case Generation for Concurrent Systems from Requirements Descriptions, In Proceedings of ICEIS, 2014.

[18] Sparx, Writing use case scenarios for model driven development, 2011. Available at http://www.sparxsystems.com

[19] M. Khandai, A. Acharya, and D. Mohapatra, A Novel Approach of Test Case Generation for Concurrent Systems Using UML Sequence Diagram, ICECT, 6, pp 157-161, 2011.

[20] H. Kim, S. Kang, J. Baik, I. Ko, Test Cases Generation from UML Activity Diagrams, in 8th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, pp. 556-561, 2007.

[21] Requirements engineering group of PUC - Rio. http://transparencia.inf.puc-rio.br/wiki/index.php

[22] Ierusalimschy, Roberto. Programming in Lua, Lua.org; 3 edition (January 3, 2013).