To cite this version: Antonelli, Leandro and Camilleri, Guy and Grigera, Julian and Hozikian, Mariangeles and Sauvage, Cécile and Zaraté, Pascale *A Modelling Approach to Generating User Acceptance Tests.* (2018) In: 4th International Conference on Decision Support Systems Technologies (ICDSST 2018), 22 May 2018 - 25 May 2018 (Heraklion, Greece).

A Modelling Approach to Generating User Acceptance Tests

Leandro Antonelli[1], Guy Camilleri[2], Julián Grigera[1,3], Mariángeles Hozikian[1], Cécile Sauvage[4], Pascale Zarate[5]

[1]LIFIA, Facultad de Informática, UNLP, Argentina
leandro.antonelli@lifia.info.unlp.edu.ar

[2]SMAC group, IRIT, 118 route de Narbonne, 31062 Toulouse Cedex 9, France
camiller@irit.fr

[3]CIC, Buenos Aires, Argentina
julian.grigera@lifia.info.unlp.edu.ar

[4]FEDACOVA, C/ Isabel la Católica 6. Pta9-10, 46004 Valencia, Spain
marian.hozikian @lifia.info.unlp.edu.ar

[5]ADRIA group, IRIT, Université de Toulouse, 2 rue du Doyen Gabriel Marty,
31042 Toulouse Cedex 9, France
zarate@irit.fr

ABSTRACT

Software testing, in particular acceptance testing, is a very important step in the development process of any application since it represents a way of matching the users' expectations with the finished product´s capabilities. Typically considered as a cumbersome activity, many efforts have been made to alleviate the burden of writing tests by, for instance, trying to generate them automatically. However, testing still remains a largely neglected step.

In this paper we propose taking advantage of existing requirement artifacts to semi-automatically generate acceptance tests. In particular, we use Scenarios, a requirement artifact used to describe business processes and requirements, and Task/Method models, a modelling approach taken from the Artificial Intelligence field. In order to generate User Acceptance tests, we propose a set of rules that allow transforming Scenarios (typically expressed in natural language), into Task/Methods that can in turn be used to generate the tests. Being high-level tests, close to the user experience, User Acceptance Tests verify that the expectations of the system are met from an end-user's point of view.

Using the proposed ideas, we show how the semi-automated generation of acceptance tests can be implemented by describing an ongoing development of a proof of concept web application designed to support the full process.

Keywords: User Acceptance Tests, Scenarios, Task/Method model, Agriculture Production Systems

INTRODUCTION

Developing software remains a complex process involving several actors and consisting of different steps. The testing step remains as one of the biggest problems, and it is frequently avoided. As a consequence, the resulting system can fail to meet users' expectations, rendering it useless. Our objective is to ease the testing step by semi-automatically generating User Acceptance Tests (UATs) from requirements artifacts. UATs represent high-level functional requirements, close to the final user's view. To do this, we combine two modelling approaches: Scenarios, from the requirement engineering field and Task/Method models, from the Artificial Intelligence field, particularly knowledge-based systems [1]. We provide rules to automatically translate scenarios to task/method models from which UATs can be generated.

This work is applied to the RUC-APS project. RUC-APS is a H2020 RISE-2015 project, aiming at Enhancing and implementing Knowledge based ICT solutions within high Risk and Uncertain Conditions for Agriculture Production Systems. In this context we will use a scenario based on agriculture production. The rest of the paper is organized as follows: we first introduce related work, then present the background introducing scenarios and the Task/method paradigm. In the third part we define our approach and in the fourth part we demonstrate a first proof of concept. Finally, we show our conclusions and future work.

RELATED WORK

The approach proposed by this paper is a problem that model-driven development has been working for a long time. This paper proposes obtaining test cases (one of the last products in software development life cycle) from requirements (one of the first). In particular, we deal with a previous artifact that can describe requirements [2].

Test cases may be generated from requirements, designs and source code. In particular, the use of abstract artifacts like UML diagrams, helps defining User Acceptance Tests, and much research has been done in this direction. Two approaches can be distinguished in this area: those that consider the relationships between elements (inter-scenario dependency), and those that consider the variations within each element (intra-scenario).

Inter-scenario dependency approaches provide a high-level organization of the artifact to cover different dependencies between them. Boucher et al. [3] transform workflow models (Use Case Maps) into Acceptance Test Cases that can be automated with the JUnit framework. Nomura et al. [4] model the business context in a matrix representing the dependency between business process, from which they design test scenarios from the perspective of Personas to cover the different situations. Sarmiento et al. [5] propose a similar approach using scenarios.

Intra-scenario approaches focus on the detail of some artifact and analyze its steps or elements to design tests. Pandit et al. [6] automatically design UATs from acceptance criteria written in the Given-When-Then template. These criteria are divided in steps, and dependencies amongst steps are arranged in a dependency graph. Lipka et al. [7] derive test scenarios from use cases stated in natural language, enriched with annotations to connect the specification with the source code of the application.

BACKGROUND

Scenarios

Scenarios can be used in different stages of software development, from clarifying business process and describing requirements, to providing the basis of acceptance tests [8]. There is a distinction between application domain (real world) and the application software (machine)[9]: during business process modelling and requirements capture, Scenarios describe events in the world, while in system specification, they describe events in the machine. Scenarios are stories about people and the activities they do to reach goals, parting from a setting and counting with resources. Their description ranges from visual (storyboards) to narrative (structured text) [10]. Leite et al. [11] propose a template with six attributes to describe Scenarios in a textual way:

- **Title**, it is the name of the scenario to identify it.
- **Goal, conditions and restrictions** to be reached after the execution of the Scenario.
- **Context, conditions and restrictions** that are satisfied and constitute the starting point of the Scenario execution.
- **Actors** and **agents** that perform actions during the Scenario to traverse the path from the context to reach the goal.
- **Resources, products** and **elements** used by the actors to perform action.
- **Episodes**: steps executed by actors using the resources in the context to reach the goal.

The text descriptions in Scenarios follow a fixed structure. In particular, episodes must be written with full sentences describing the subject, the action they perform, and if necessary the resource used. The following example describes a Scenario for farmer packing products. The example also includes the cases to consider for testing the scenario. These test cases do not belong to the original structure of the scenario:

Scenario: Packing the products
Goal: Put the products in boxes in order to distribute them
Context: The products have recently been harvested
Resources: Products, Box
Actors: Farmer

Episodes:	The farmer washes the products
	The farmer brushes the products
	The farmer determines the destination of the products
	The farmer determines the quality levels of the products according to the destination
	The farmer determines the appropriate box according to the destination
	The farmer chooses the products that satisfy the quality levels
	The farmer packs the chosen products in the box
Test cases:	Temperature forecast obtained / not obtained
	Sun radiation forecast obtained / not obtained
	Rain forecast obtained / not obtained
	There is no best date to plant

Task/Method Paradigm

The task/method paradigm is a knowledge modelling paradigm (mainly from the artificial intelligence field [12], [13]) that sees reasoning as a task. Knowledge is expressed in a declarative way, making it easy to process by execution engines or planners [1]. A task/method model is composed by a **domain model** and a **reasoning model**. The former describes the objects of the world being used (directly or indirectly) by the latter, similarly to an application ontology. It is often described in UML language and implemented with OO languages. The reasoning model describes how a task can be performed. It uses two modelling primitives:

1. **Task**: it is a transition between two world state families (an action) and is defined by the following fields: *Name, Par, Objective* and *Methods*.

2. **Method**: it describes one way of performing a task. A method must generate a state to satisfy its task's objective, although different methods could produce different effects. It is characterized by the following fields: *Heading, Prec, Effects, Control* and *Subtask*.

The task's field *Name* specifies the name of the task. The field *Par* contains the list of parameters, that is, all objects handled by the task. For example, in a task *Pack*, the parameter list could be *(farmer, products)* which are domain objects (from domain model) used by the task *Pack*. We will write *Pack(farmer, products)*. The list of methods which can be applied to perform a task is described in the field *Methods*. A terminal task is a directly executable task, with only one method to perform it. The method's field *Prec* contains conditions that must be satisfied to apply the method. The execution order of subtasks is described in the *Control* field, and sub-tasks are recorded in the *Subtask* field. Note that, by essence, Task/Method models are hierarchical. Here we explained only the fields used in this work, see [1] for a full reference.

APPROACH

The proposed approach consists in representing scenarios in the form of Task/Method models. Being a modelling paradigm, the building of Task/Method models requires modelling effort. On one hand, the integration of this modelling activity during the definition of scenarios facilitates an early identification of misunderstandings between stakeholders. Moreover, as Task/Method models are operational models, they can be executed to generate test cases. On the other hand, building a task/method model at early stages shouldn't take much effort. To reduce this effort, we propose a semi-automatic translation of scenarios to task/method models. We use scenarios expressed in natural language, since it's the natural way to describe them.

In the packing example presented previously, the translation process would produce the following Task/Method model for the general scenario and the first subscenario, respectively. The strategy to obtain UATs consists in determining two possible situations for each action, one in which the action can be performed successfully, and another where the action failed.

```
Method: M1
Task: Packing(products)
Control:  wash(farmer, products);
          brush(farmer, products);
          determines(farmer, destination, products);
          determines(farmer,quality_levels,products,destination);
          determine(farmer, appropriate_box, destination);
          choose(farmer, products, quality_level);
          pack(farmer,products,box);
```

```
Method:  M21
Task: brush(farmer,products) (IR2b)
Precondition:
  not Product_correctly_washed
Control:
  message("not correctly washed, stop");
    stop;
```

In the next subsection, we present the translation rules, and a proof of concept.

Translation Rules

The translation of scenarios to task/method is performed thanks to the following rules:

Rule 1. Tasks Identification: each verb in the Scenario's episodes is translated into a task in Task/Method model. Each Scenario title is also a task in Task/Method model. Examples:

Episode: The farmer washes the products → Task: Wash
Episode: The farmer brushes the products → Task: Brush
Episode title: Packing the products → Task: Pack

Rule 2. Task's Parameters Identification: each actor and resource used in the episodes of a Scenario is translated by a parameter in Task / Method model. Examples:

Episode: The farmer wash the products → Task: Wash(farmer, product)
Episode: The farmer brush the products → Task: Brush(farmer, product)

Rule 3. Episode's method: the episodes part of a scenario is translated by a method in Task / Method model. Examples:

```
Episodes:        The farmer wash the products
                 … → Method: M1
```

Rule 4. Sequence of tasks: the sequence of different lines in the episodes part of a Scenario determines the sequence of tasks in the control part of a method in the Task / Method model. The use of expressions like "then", "after", etc... in the episodes of a Scenario determines also a sequence of tasks in the method's control part. Examples:

```
Episodes:
  The farmer washes the products      or    The farmer washes the products, then brushes the products
  The farmer brushes the products

Method: M1
Control: wash(farmer, product); brush(farmer, product)
```

Rule 5. Test Case Method: In this work, we assume that each test case (Test cases part of scenario) corresponds to the achievement status (succeed or fail) of a task. In a failure situation, the scenario will stop. This stop case will be represented by a method for the next task in which the precondition field correspond to the test case failure. For example:

```
Test case: Temperature forecast obtained / not obtained →
Method:  M21
Task:  bush(farmer,products) # next task
precondition: not Product_correctly_washed
Control:  message("not correctly washed, stop"); stop;
```

The natural language used in the expression of scenarios is limited, we also assume that episodes part only contains a "nominal" way to achieve a scenario, i.e. we consider that execution of every task succeeds. In the test cases part, only failures of task achievement are considered. With these assumptions and a few translation rules, it is possible to automatically translate scenarios such as the packing scenario. Of course, this automatic translation has to be used as a preliminary design and it should be analyzed and enriched by a test case designer.

PROOF OF CONCEPT

We are developing a web application where users can describe Scenarios and obtain the Task/Method model that implements the UATs. Figure 1 depicts the architecture: a client module allows describing the Scenarios and see the derived Task/Method model. The server module performs the derivation using a Natural Language Processor and a set of rules. Rules determine scenarios' processing, elements to be identified, and how they relate to produce the Task/Method. The NLP processes Scenarios, obtaining the elements determined by the rules.

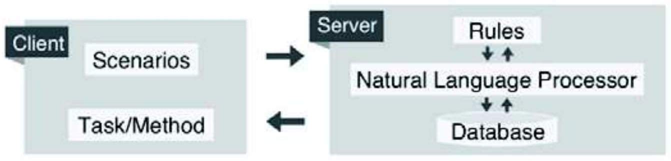

Figure 1. Architecture of the application

The system is being implemented with Node.js (https://nodejs.org) and Angular 2 (https://angular.io). This will allow users to define the scenarios and obtain the Task/Method executing the rules specified through the Stanford Natural Language Processing Framework (https://nlp.stanford.edu). After that, the final translation will be shown to the client.

CONCLUSION

In this paper we have shown a new way of generating acceptance tests from well-known requirements artifacts, by presenting a set of rules to guide the implementation of semi-automated solutions and shown the first steps towards a supporting tool. We are now working in completing the ruleset, by adding the rules required to translate iterative episodes into tasks. For example, each verb used in the episode of a Scenario that describes iteration should be written as a while expression in in Task/Method model: while <condition> <block>. This would help to support other scenarios that require iterative tasks, e.g. "For each order, determine the time needed to take the products to the destination". We also plan to publish the web application in order to experiment with the presented ideas in real development settings, so we can assess the benefits of semi-automatically generated UATs.

ACKNOWLEDGEMENTS

Authors of this publication acknowledge the contribution of the Project 691249, RUC-APS: Enhancing and implementing Knowledge based ICT solutions within high Risk and Uncertain Conditions for Agriculture Production Systems (www.ruc-aps.eu), funded by the European Union under their funding scheme H2020-MSCA-RISE-2015

REFERENCES

[1] G. Camilleri, J.-L. Soubie, and J. Zalaket, "TMMT: Tool Supporting Knowledge Modelling," in *Knowledge-Based Intelligent Information and Engineering Systems*, vol. 2773, 2003, pp. 45–52.

[2] V. A. Rubin, A. A. Mitsyuk, I. A. Lomazova, and W. M. P. van der Aalst, "Process mining can be applied to software too!," in *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement - ESEM '14*, 2014, pp. 1–8.

[3] M. Boucher and G. Mussbacher, "Transforming Workflow Models into Automated End-to-End Acceptance Test Cases," in *Proceedings - 2017 IEEE/ACM 9th International Workshop on Modelling in Software Engineering, MiSE 2017*, 2017, pp. 68–74.

[4] N. Nomura, Y. Kikushima, and M. Aoyama, "A Test Scenario Design Methodology Based on Business Context Modeling and Its Evaluation," *2014 21st Asia-Pacific Softw. Eng. Conf.*, vol. 1, pp. 3–10, 2014.

[5] E. Sarmiento, J. C. S. P. Leite, E. Almentero, and G. Sotomayor Alzamora, "Test Scenario Generation from Natural Language Requirements Descriptions based on Petri-Nets," *Electron. Notes Theor. Comput. Sci.*, vol. 329, pp. 123–148, 2016.

[6] P. Pandit, S. Tahiliani, and M. Sharma, "Distributed agile: Component-based user acceptance testing," in *2016 Symposium on Colossal Data Analysis and Networking (CDAN)*, 2016, pp. 1–9.

[7] R. Lipka, T. Potuak, P. Brada, P. Hnetynka, and J. Vinarek, "A Method for Semi-Automated Generation of Test Scenarios Based on Use Cases," in *Proceedings - 41st Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2015*, 2015, pp. 241–244.

[8] I. Alexander and N. Maiden, "Scenarios, Stories, and Use Cases: The Modern Basis for System Development," *IEEE Comput. Control Eng.*, vol. 15, no. 5, pp. 24–29, 2004.

[9] M. Jackson, "The world and the machine," in *Proceedings of the 17th international conference on Software engineering - ICSE '95*, 1995, pp. 283–292.

[10] R. Young, *The requirements engineering handbook*. 2004.

[11] A. Hussain *et al.*, "Review on formalizing use cases and scenarios: Scenario based

testing," *2015 Int. Conf. Emerg. Technol.*, vol. 3, no. 3, p. 1, 2015.

[12] F. Trichet and P. Tchounikine, "DSTM: A framework to operationalise and refine a problem solving method modeled in terms of tasks and methods," *Expert Syst. Appl.*, vol. 16, no. 2, pp. 105–120, 1999.

[13] G. Schreiber, H. Akkermans, A. Anjewierden, R. De Hoog, N. R. Shadbolt, and B. Wielinga, *Knowledge Engineering and Management: The CommonKADS Methodology*, vol. 99. 2000.