Test Cases from Functional Requirements Using Model Transformations

Javier J. Gutiérrez, María J. Escalona, Manuel Mejías, Isabel Ramos
Computer and Languages Department, University of Sevilla.
Avd. Reina Mercedes sn. 41012
Sevilla, Spain
{javierj, escalona, risoto, iramos}@us.es

Abstract:
This abstract describes the experiences and ongoing work applying MDE to the generation of test cases from functional requirements.

1. Introduction

Nowadays, one of the main research files is model-driven engineering (MDE). The main goal of MDE is to define a set of metamodels and a set of transformation among models (instances of the metamodels). From this point of view, MDE is not only a valuable research field but a useful tool for resolve problems in other domains. One of those domains is the test field. Generation of test cases systematically implies the need for defining the input information, the output information and the automate process for transforming from the input to the output. Input and output and transformations may be defined with MDE tools. This abstract describes de efforts of the authors to apply MDE to their work about generation of test cases from functional requirements. Section 2 introduces a brief description of the transformation process used in this abstract. Section 3 describes the metamodels needed for functional requirements (input information) test cases (output information) and introduces an example of transformation. Finally section 4 exposes conclusions and acknowledges.

2. An overview of the transformation process

The process of generate test cases from functional requirements has been described in previous papers of the same authors [4][5][6]. However, none of them uses a MDE approach. This section introduces a brief description of this process, which be formalized in metamodels and transformations in next section.

One of the main techniques for extracting test cases from a functional requirement is the scenario analysis. The main goal is to identify scenarios from the functional requirement and propose them as text cases. A functional requirement describes an interaction among system and actor using steps. Generally, the functional requirement includes steps to manage alternatives o erroneous situations. Thus, a scenario may be defined as a concrete set of steps from the functional requirement, this means, a sequence of steps with no alternatives or errors. This technique is used, for example, in [1][2][8] and [9].

3. Generating test cases from a MDE perspective

This section describes an overview of this elements applied to the generation of test cases from functional requirements. As seen before, three elements are needed to apply a MDE approach: source metamodel, target metamodel and transformations. Next sections introduce these elements.

3.1. Functional Requirements Metamodel

Source metamodel must be a functional requirements metamodel. There are many approaches to define textual requirements (for example [3] and [2]). However the majority of the existing approaches are based on a common core of elements. These elements include the participants, preconditions, post-conditions, a set of main steps (to achieve the goal of the use case) and a set of exceptional steps (to manage alternative and erroneous scenarios). These common elements have been extracted from the cited approaches and modeled in the functional requirement metamodel introduced in figure 1 and briefly described in the following paragraphs.

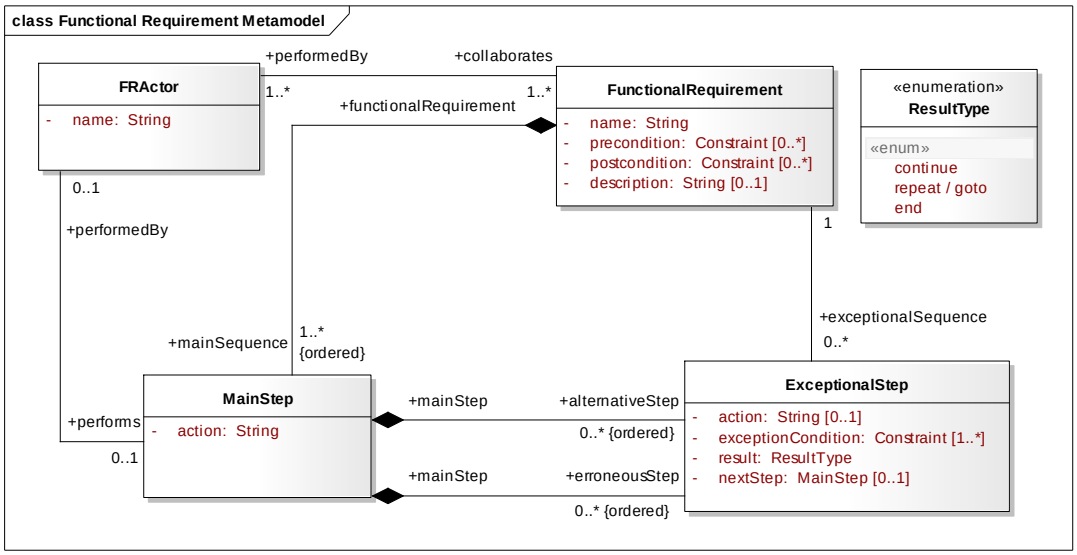

Figure 1. Functional Requirements Metamodel.

A *FRActor* element models an external actor who participates in the functional requirement. The *FunctionalRequirement* element defines an interaction among the system and a set of external actors. The behavior of the *FunctionalRequirement* metamodel is defined with preconditions, postconditions and a sequence of steps, called main steps. This sequence defines the steps performed by actors and system to achieve the goal of the functional requirement. The alternative steps indicate variants or additional behavior for the main steps. The erroneous steps indicate error scenarios after performing main steps. The *exceptionCondition* attribute indicates a boolean expression that must be true for performing the step and the result attribute indicates the behavior at the end of the exceptional step. Three results have been defined as the enumerated *ResultType* element: end the execution, continue the execution and repeat/go-to to another step. A continue result indicate that the execution of the functional requirement continues with the next step after the ending of the alternative or erroneous step. The repeat or go-to result indicates that the execution of the use case continues executing the step indicated in the *nextStep* attribute and the end result indicates than the use case ends. The *nextStep* attribute is only mandatory when the result type is repeat / go-to.

No concrete syntax is defined in this paper for instances of this metamodel (this means, for functional requirements). For example, the textual tabulation or activities diagrams, as introduced in previous papers of the author, like [7].

3.2. Test Cases Metamodel

Target metamodel must allow to define test cases as the scenarios from the functional requirements (as defined in previous section). This target metamodel is showed in figure 2.

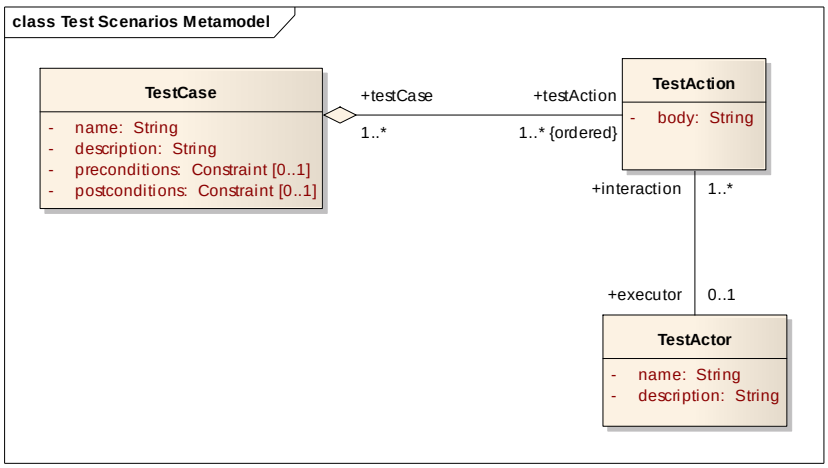

Figure 2. Test Scenarios Metamodel.

Elements in target metamodel are quite similar to the elements of source metamodels. A *TestCase* element is a scenario from a functional requirement. The steps of a test case are the elements *TestActions*. As mentioned before, there are no alternatives in the set of steps. Finally, some test steps will be execute for the system and will be a valuable point to include assertions to verify the successful execution of the test case, but other steps must be executed for an external element: a human, a test harness, other subsystem, etc. This fact is modeled with the *TestActor* element.

Again, there are no definitions of concrete syntaxes, so test cases may be represented as tabular text (in the same way that functional requirements in previous section) or using the UML diagrams proposed in the UML Testing Profile [10].

3.3. Transformations between models

As seen before, the last element for an MDE is a set of transformations between instances of the source metamodel and instances of the target metamodel. An example of transformation to create a new *TestCase* element from a *FunctionalRequirement* element is described in next paragraph. This transformation has been defined using the QVT-Relational language, which is the transformation language defined by the OMG [11].

```
top relation FunctionalRequirement2TestCase {
  checkonly domain functionalRequirements rf: FunctionalRequirement
  { name = nrf,           description = com
    precondition = pre,   postcondition = pos
  };
  enforce domain testScenarios e: TestCase {
    name = nrf,           description = 'Test case for '+nrf,
    precondition = pre,   postcondition = pos
  };
}
```

This QVT transformation is quite simple due there is no need to change the information from the functional requirements. However, other transformations are more sophisticated, for example the set of transformations dedicated to traverse all the execution paths of the functional requirement to build the scenarios. This set is still an ongoing work.

4. Conclusions and acknowledges

This abstract has briefly presented an approach for test case generation using MDE philosophy. There are two main benefits of using MDE philosophy to define test case generation process. First, MDE adds a set of tools for formalizing the algorithms and information structures presented in any work about test case generation. Second, due the strong relation between MDE and UML, several generic supporting tools like UML Profiling modelling tools, etc are already available.

This work is supported by the Ministry of Science and Education (Spain) under the National Program for Researching, Development and Innovation, project QSimTec (TIN2007-67843-C06-03) and REPRIS (TIN2005-24792-E).

References

[1] Boddu R., Guo L., Mukhopadhyay S. 2004. RETNA: From Requirements to Testing in Natural Way. 12th IEEE International Requirements Engineering RE'04.

[2] Cockburn, A. 2000. Writing Effective Use Cases. Addison-Wesley 1st edition. USA.

[3] Escalona M.J. 2004. Models and Techniques for the Specification and Analysis of Navigation in Software Systems. Ph. European Thesis. Department of Computer Language and Systems. University of Seville. Seville.

[4] Gutiérrez J.J. Escalona M.J. Mejías M. Torres J. 2006. Derivation of test objectives automatically. Fifteenth International Conference On Information Systems Development (ISD06). Budapest, Hungary, 31 August – 2 September, 2006

[5] Gutiérrez J.J. Escalona M.J. Mejías M. Torres J. 2006. Towards a Complete Approach to Generate System Test Cases. ICEIS Doctoral Consortium. Oaphos, Cyprus.

[6] Gutiérrez J.J. Escalona M.J. Mejías M. Torres J. 2006. Modelos Y Algoritmos Para La Generación De Objetivos De Prueba. Jornadas sobre Ingeniería del Software y Bases de Datos JISBD. Sitges. Spain.

[7] Gutierrez, J.J., Nebut, C., Escalona, M.J., Mejías, M., Ramos, I. Visualization of use cases through automatically generated activity diagrams. Lecture Notes in Computer Science. 5301. pp. 83-96. 2008

[8] Heumann , J. 2002. Generating Test Cases from Use Cases. Journal of Software Testing Professionals. EEUU.

[9] Naresh, A. 2002. Testing From Use Cases Using Path Analysis Technique. International Conference On Software Testing Analysis & Review. EEUU

[10] Object Management Group. 2003. The UML Testing Profile. www.omg.org.

[11] Query QVT-Merge Group, Revised submission for MOF 2.0 Query/Views/ Trans-formations RFP. 2004, Object Management Group, http://www.omg.org/cgi-bin/apps/doc?ad/04-04-01.pdf.