ImprovingNDT with Automatic Test Case Generation

J. Gutiérrez , M. J. Escalona , M. Mejías , F. Domínguez ,and C. R. Cutilla

Abstract

The model-driven development defines the software development process as a set of iterations to create models and a set of transformations to obtain new models. From this point of view, this paper presents the enhancement of a model-driven approach, called navigational development techniques (NDT), by means of new models and transformations in order to generate test cases. It also states some conclusions from the research work and practical cases in which this approach was used.

1 Introduction

Model-driven development (MDD hereinafter) is a software engineering paradigm focused on creating and exploiting domain models [1]. Navigational development techniques (NDT) [2] is a development framework which follows MDD. Therefore, NDT describes the full software development cycle indicating which models generate in each phase and how to derive the models of a phase from those of a previous phase. Projects developed with NDT start with a goal-oriented phase of require-ments and apply use cases for defi ning requirements and transformations so as to generate the following models.

NDT was initially defi ned to deal with Web development requirements, but it has evolved in the last years and nowadays it offers a complete support for the complete life cycle. NDT covers viability study, requirements treatment, analysis, design, construction, or implementation as well as maintenance and test processes. Additionally, it supports a set of processes to bear out project management and quality assurance.

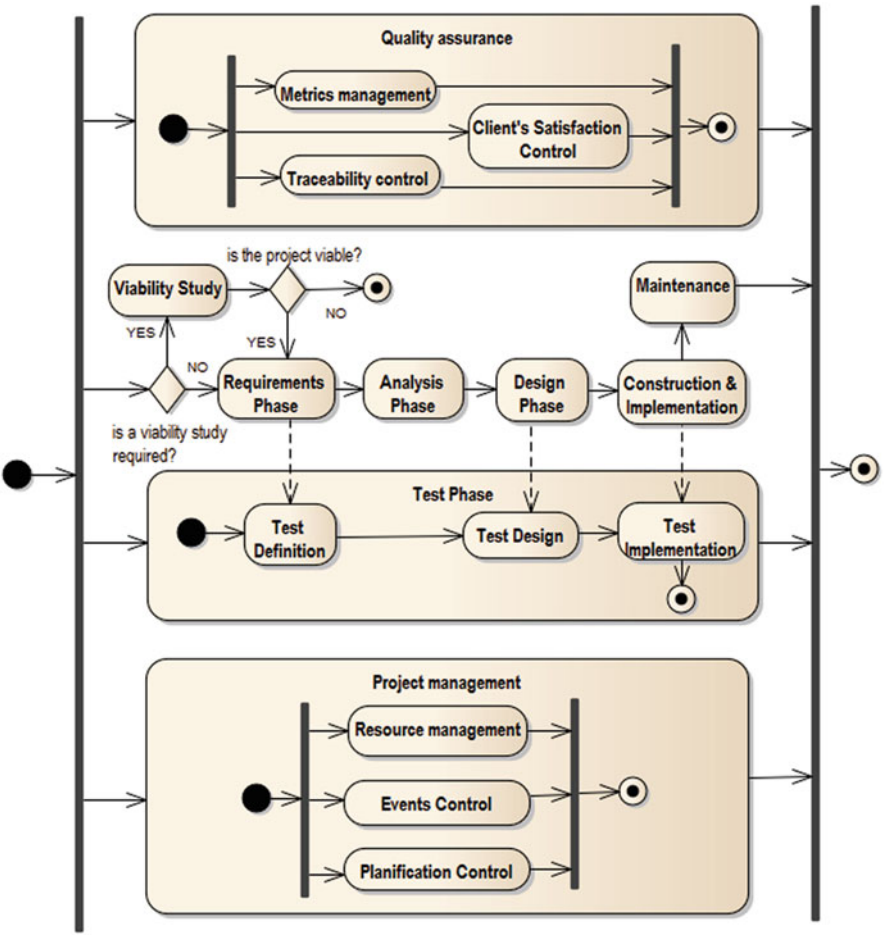

Fig. 1 NDT development process overview

Figure 1 depicts the development process as defined in NDT. This development process is independent from the development life cycle, so any life cycles (e.g., cascade or iterative) may be applied. Testing is always a mandatory task (Fig. 1) regardless of the type of project. Thus, there is a need of incorporating techniques in order to define test cases. Moreover, due to the MDD nature of NDT, these techniques should be described in terms of the proper elements of MDD, mainly models and transformations from models to models.

This paper describes how NDT has been extended to incorporate functional system test cases [3]. These test cases verify that the system under test commits the behavior defined in its functional requirement. NDT models the functional requirements as use cases; thus, both terms will be used as synonyms in this paper.

This paper is organized as follows. After this introduction, Sect. 2 offers an overview of the existing techniques dealing with generating functional test cases from functional system requirements defined as use cases. Section 3 describes the techniques used to generate test cases. Section 4 summarizes the extension of NDT to incorporate those techniques from a MDD perspective. Section 5 presents practical applications for NDT enrichment with test case generation. Finally, Sect. 6 states the conclusions and ongoing work.

2 Related Work

There are several approaches generating functional test cases specifically from a functional requirements model defined as use cases by means of MDD. A survey about this issue, which updates the original survey published in [4], has been published in [5] at the end of 2011. Some specific approaches studied in Escalona's survey are described in next paragraphs.

Frölich et al. [6] introduce an approach describing how to translate a functional requirement from natural language into a state-chart diagram in a systematic way as well as how to generate a set of functional test cases from that diagram. Naresh [7] presents an approach dealing with translating a functional requirement from natural language into a flow diagram and performing a path coverage technique to generate test cases. Mogyorodi [8] introduces an approach describing functional requirements as cause-effect graphs which generates test cases from diagrams. Boddu et al. [9] present an approach divided into two blocks: the first one describes a natural language analyzer generating a state machine from functional requirements, and the second one shows how to create test cases from such state machine.

Ruder's [10] approach starts with functional requirements written in natural language. The result is a set of functional test cases obtained from a coverage criterion based on combinations that support Boolean propositions. Binder's book [11] describes the application of the category-partition method over use cases. The categories are any point in which the behavior of the use case may be different between two realizations of the use case. This application is named the extended use case pattern. Finally, Ibrahim et al. [12] offer a tool, called GenTCase, which generates test cases automatically from a use case diagram enriched with each use case tabular text description.

Escalona's survey claims that there is no definitive approach that closes the problem of generating functional text cases automatically in a satisfactory way, what implies a lack of evolution among the existing approaches. Thus, there are some aspects to be improved, like the use of standards for inputs and outputs, the application of standards and more formal methods to describe the process itself, the need for empirical results or the measure of the possible automation, and a profitable tool supporting, among others. Conclusions of Denger's survey go in the same line.

3 Techniques for Test Cases Generation

Two techniques have been identified for generating test cases from use cases, from the surveys cited in previous sections: round-strip strategy and extended use cases (terminology defined by Binder in [11]). Below, these techniques will be described in depth.

The round-strip strategy consists in the application of a classic algorithm of path finding over a state machine. The behavior described in a functional requirement may be managed as a graph or as state machine despite its concrete syntax. Hence, a path searching allows identifying all the different paths across the behavior. Each path will be a scenario designed together with the system. Each scenario is a potential test case for testing the right implementation of such scenario in the system under test. Generation of test cases from state machines is a widely described topic in research literature. Previous section presented several references about this topic in the specific use cases context, like [6, 7, 9]. Figure 2(a) shows an example of the round-strip strategy using the behavior of a use case defined as an activity diagram.

The extended use case pattern consists in applying the category-partition method [13] to use cases. The category-partition method is a technique based on identifying categories and partitions and then generating combinations among such partitions (Fig. 2b). In the context of functional requirements, a category is any point for which the functional requirement defines an alternative behavior (Fig. 2b). Besides, a partition is defined as a subset of the domain of the condition evaluated in the category which decides whether a concrete piece of behavior is executed or not. Once all categories and partitions are identified, a combination among them is performed and each combination becomes a potential test case. The previous section presented several references about this topic in the specific context of use cases, like [10] or [11]. Figure 2b shows an example of the category-partition method (as described in [11]) using the same behavior as Fig. 2a.

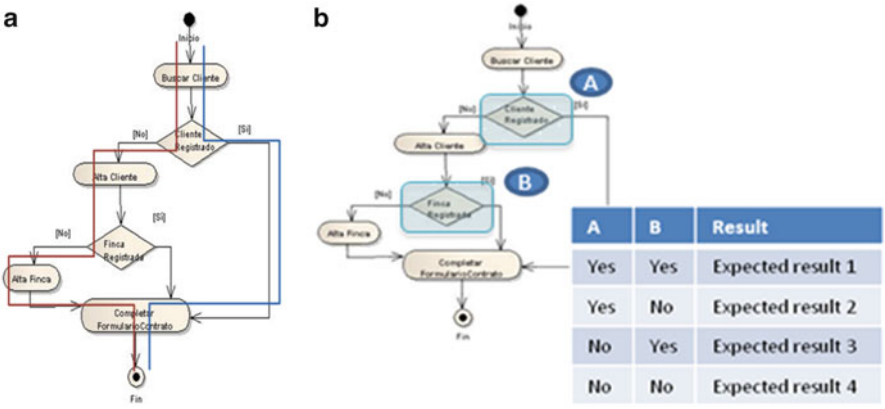

Fig. 2 Examples of round-strip strategy (**a**) and extended use cases (**b**) techniques

4 Extension of NDT

This section describes the work carried out to extend NDT after including the two techniques presented in the previous section. Section 4.1 defines the information involved. Then, Sect. 4.2 defines how to apply both techniques to obtain the target test artifacts from the functional requirements. Finally, Sect. 4.3 offers an overview on the application of Sects. 4.1 and 4.2 results.

4.1 Concepts and Metamodels

Due to the model-driven nature of NDT, the concepts involved in generating functional test cases should be identified and defined as metamodels. A metamodel defines the concept in terms of its attributes and its relationships with other concepts [1].

Four metamodels were designed. The first one (Fig. 3) defines the necessary elements from functional requirements to generate test cases. These elements constitute a subset of functional requirements. Therefore, it only involves the elements used for test cases generation. This metamodel may be applied with other frameworks apart from NDT. The functional requirement metamodel (Fig. 3) includes classic elements of functional requirement defined as *use cases*, Step or Actor, among others, widely described in the literature.

The second metamodel (Fig. 4) defines the concepts resulting from the round-strip technique (Fig. 2a). Each path is called test scenario (element *TestScenario* in Fig. 4) and the traverse/crossed steps are classified into actions (element *ActionFromTestScenario* in Fig. 4) when performed by an external actor or into verifications (element *VerificationFromTestScenario* in Fig. 4) when performed by the system and, therefore, is suitable to introduce an assert during the test.

The third metamodel (Fig. 5) defines the concepts resulting from the category-partition method. Categories are modeled using the element *OperationalVariable*

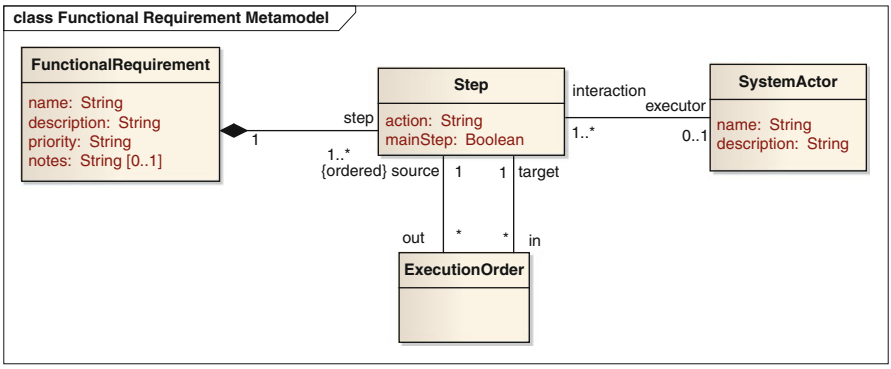

Fig. 3 Metamodel for functional requirements

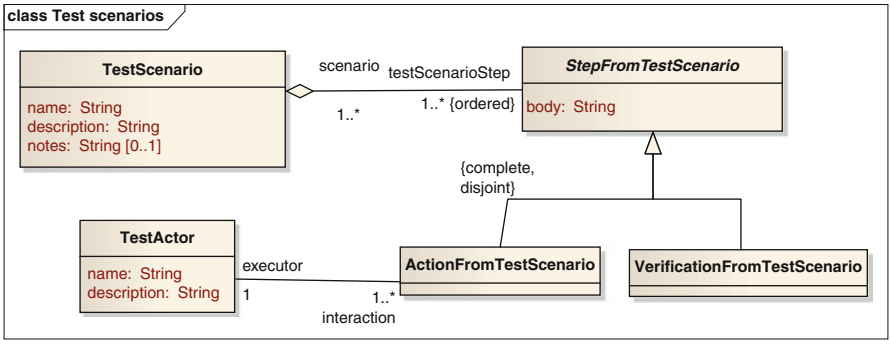

Fig. 4 Metamodel for test scenarios

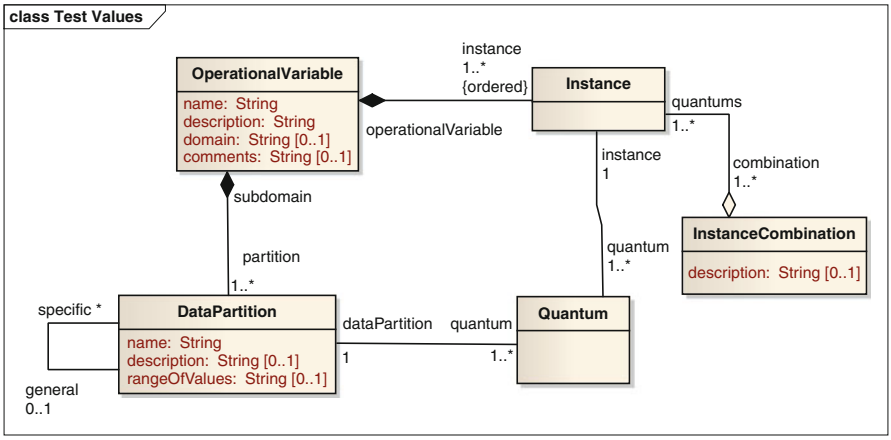

Fig. 5 Metamodel for test values

(as named in [11]), whereas partitions are modeled through the element *Partition*. The element *Instance* points out an evaluation of an operational variable, for example, A or B cells in Fig. 2b, and allows distinguishing it from other evaluations of the same operational variable, in case the behavior of the functional requirement has loops. A *Quantum* element models a value transfer from a partition to an instance. A combination (a row in Fig. 2b) is modeled using the element *Instancecombination*.

Finally, the last metamodel introduces artifacts that combine the results of the two previous techniques in the same model. This last metamodel does not introduce any new information. However, it offers glue elements to represent the information through a common artifact (called test case), the steps from a functional requirement as well as a combination of partitions. Figure 6 shows the tracing relation between the four metamodels. Tracing enables knowing which test artifacts have been generated for each functional requirement.

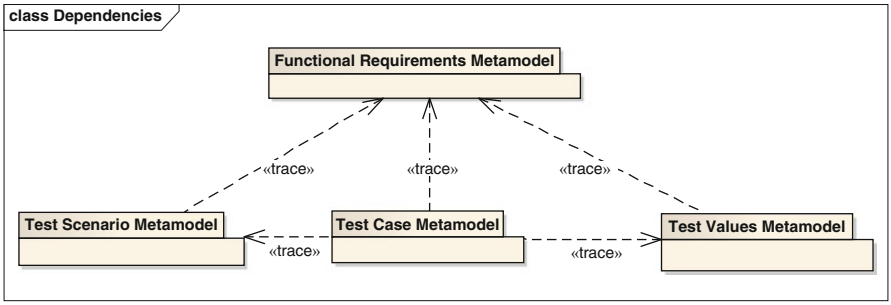

Fig. 6 Tracing relationships among metamodels

Some additional elements from the metamodels have been omitted. These elements introduce additional concepts like preconditions and packages. The four former metamodels have been added to the set of metamodels managed and supported by NDT as part of its MDD development process.

4.2 Relations and Transformations

Section 4.1 described the concepts involved in the improvement of NDT to generate functional system test cases. This section goes one step beyond and describes how to apply the two techniques presented in Sect. 2 (round-strip and extended use cases) using the information from the functional requirements metamodels (in the previous section) as source and the information from the testing metamodels as target.

The process of applying both techniques is defined according to the identification of a set of relations between source concepts (functional requirements) and target concepts (test scenarios and operational variables combinations), as observed in Fig. 7. The task of identifying these relations consists in detecting how to build up one target element, for example, a test case, by means of the source elements and their information. Next paragraphs provide an overview of the three relations (named T1, T2, and T3 in Fig. 7) defined to create test scenarios, combinations of operational variables, and test cases from functional requirements.

Relation T1 involves functional requirements and the round-strip strategy. As it was represented in Fig. 2a, the functional requirement behavior may be modeled as a state machine; the concept *Step* from Fig. 3 models the states; and the concept *Execution Order* models the transitions. Thus, a classic coverage criterion may be selected to traverse/cross the functional requirement and generate test scenarios. The all-loops criterion, in which all combinations among loops are traversed at least once, is the one selected to extend NDT. Test scenarios steps are generated from all the functional requirements steps. Action (element *ActionFromTestScenario*) and verification (element *VerificationFromTestScenario*) classifications depend on whether there is a relation with a system actor. Finally, test actors are generated from actors, which, due to their attributes, are the same ones.

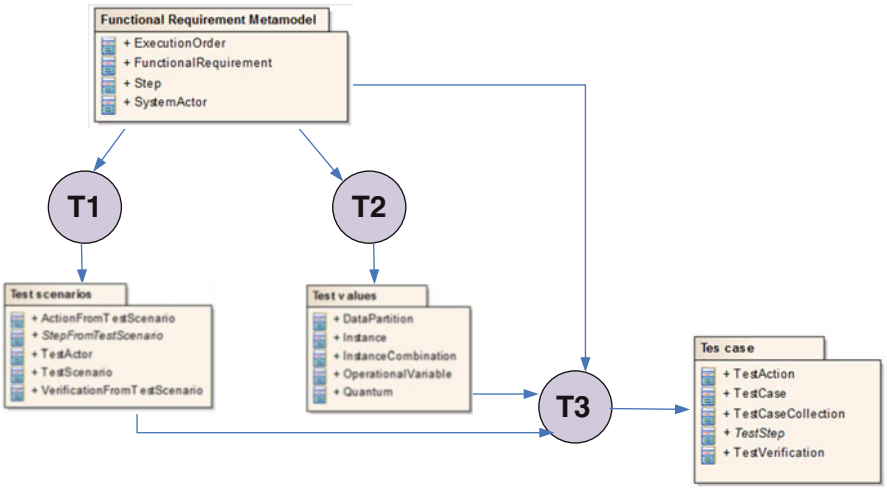

Fig. 7 Transformations among models

Relation T2 in Fig. 7 involves functional requirements and the category-partition method. Operational variables are created from steps that have more than one output transition (modeled as an *ExecutionOrder* element). The outputs of the steps generate the different partition. Again, combinations may be calculated using several criteria, from calculating all possible solutions or calculate just a subset.

Relation T3 (Fig. 7) combines both techniques results. Test scenarios and combinations of operational variables merge using test cases.

4.3 Application

On one hand, previous metamodels and transformations do not impose a concrete representation of the involved elements (functional requirements, test scenarios, operational variables, and test cases), but on the other hand, working directly with the metamodels object may be difficult, as shown in Figs. 8 and 9. NDT does not impose a concrete syntax for requirements, allowing the definition of use cases by means of either a model defined in UML or a text template. As it can be observed in Fig. 8, several concrete syntaxes may be used for defining functional requirements. The "?" indicates that any other syntaxes or formats plus the indicated one may be used.

Thus, the first step to apply the generation of functional test cases from functional requirements deals with defining a process for extracting a functional requirement model in accordance with the metamodel introduced in Sect. 4.1. This process depends on the specific syntax and it is out of the scope of this paper. Some previous work in this line was published in [14].

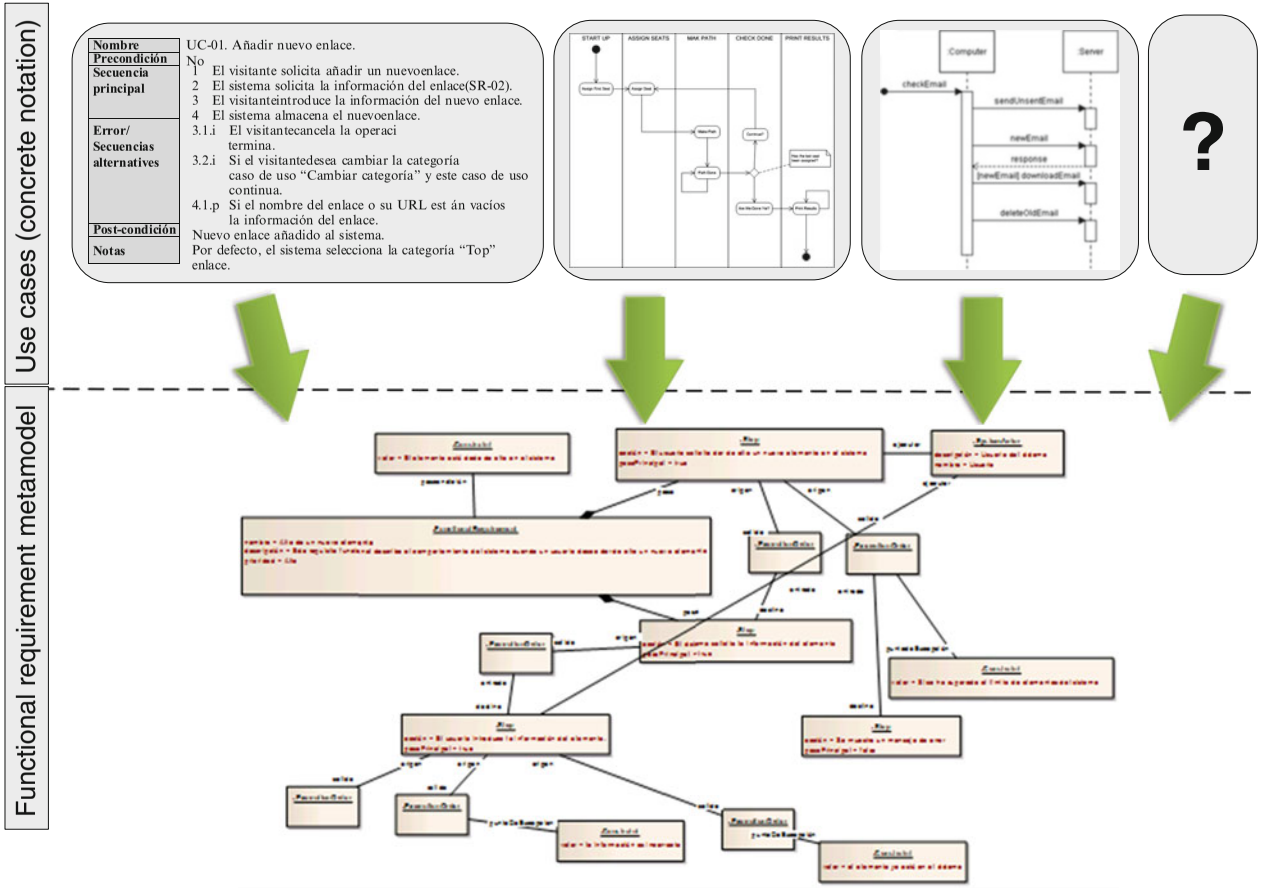

Fig. 8 Concrete syntaxes for functional requirements model

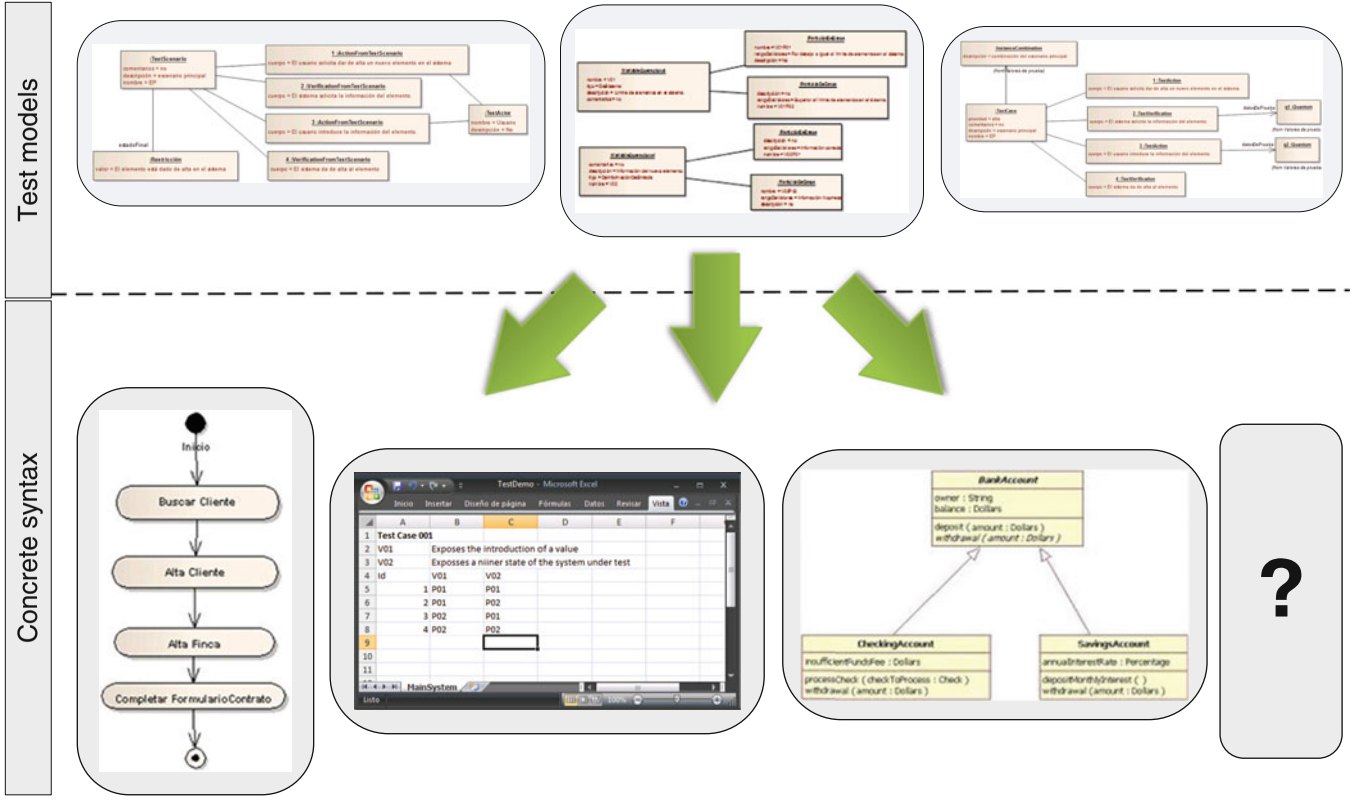

Fig. 9 Concrete syntaxes for test artifacts

Table 1 Metrics for QVT-operational code

	T0	T1	T2	T3
Total lines	124	118	290	170
Lines of codes	104	97	238	124
No. of mappings	1	4	5	3
No. of helpers	1	2	3	1
No. of queries	3	2	1	3
No. of input models	1	1	1	3
No. of output models	1	1	1	1

In the same way, a model including the testing artifacts obtained after the applications of both previous techniques may not result the most suitable syntax (Fig. 9). Even more, different models may require different syntaxes. For example, a valid syntax for a test scenario model (an activity diagram) may not be the proper syntax for instance combinations. In this case, a table or an Excel sheet (Fig. 9) should be more valuable. Again, a question mark in Fig. 9 represents any other valuable syntax. Next section introduces the software tools that implement these techniques and explain the concrete syntax they manage.

The relations stated in the previous section (T1, T2, and T3 from Fig. 7) were defined through the QVT-operational language as a necessary step to know how to implement the transformation process into an automatic tool. The QVT code may be downloaded from [15]. The metrics of the QVT code are collected in Table 1 and defined in [16].

Table 1 adds an additional transformation, called T0, not included in Fig. 7. This transformation contains common code used in other transformations. As reference, the Umls2Rdb transformation written in QVT operational and included in the QVT reference [17] has 65 lines of code, 6 mappings, and 1 query.

5 Practical Experiences

Nowadays, several companies in Spain work with NDT. This is possible due to the fact that NDT is completely supported by a set of free tools, mainly grouped in NDT-Suite [18]. This suite enables the definition and use of every process and task supported by NDT (Fig. 1) and offers relevant resources for quality assurance, management, and metrics with the aim of developing software projects. The suite was also extended to implement the first technique for test case generation using activity diagrams as the concrete input for functional requirements, and for the concrete syntax of the test scenarios generated. The implementation of the second technique is still an ongoing work.

However, the MDD perspective allows the concrete notations independency. Thus, the metamodels and transformations defined in previous section may be used out of the scope of NDT. The only request is that the source functional requirements must include the concepts defined in the functional requirements metamodel used

as the basis for the process. To remark this independency, a second tool, called MDETest, was created. The main differences between this tool and NDT-Suite are that MDETest implements the three target metamodels and it generates the tool use instances only for metamodels, so that it does not impose any restrictions over the concrete notations of the functional requirements input. Nowadays, this tool supports activity diagrams such as the syntax for functional requirements whereas it does not support any concrete syntax for the output. This tool is also available in [15].

A very first application of this extension was the AQUA-WS [19] project. EMASESA is a public company which deals with the general management of the urban water cycle, providing and ensuring water supply to all citizens in Sevilla. AQUA-Web-Services (also called AQUA-WS) project consists in the development and implantation of an integrated business system for customer management, interventions in water distribution, cleaning, and net management. This system had 1,808 functional requirements, which individually include several scenarios and alternatives.

During the development of AQUA-WS project the development team used NDT-Tool to generate the test plan, which had over 7,000 test cases generated from the different scenarios out of the 1,808 functional requirements. Estimating 5-min length to create a test scenario in the modeling tool, the amount of time gained with NDT-Tool reached 583 h (73 days, working 8 h a week). Even more, the test cases obtained were classified in the right packages and they had tracing relations with the use cases source. The modeling tool used to manage use cases and test cases has search options to map the tracing relations, which makes more easy the task of working with a wide set of test cases.

6 Conclusions and Ongoing Work

This paper presents a model-driven process, based on metamodels and transformations, with the aim of generating test cases from functional requirements. As a result of this work, NDT has been enriched with metamodels and transformations so as to generate test cases from functional requirements automatically by means of the NDT-Suite tool.

Extension has been tested in several projects and it opens new research lines. Firstly, we have to work in test cases prioritization mechanisms, consisting in giving relevance to functional requirements, as well as in redundant test cases detection. The practice concludes that it continues producing a high number of redundant test cases that the test teams have to detect by hand. One last ongoing work would deal with supporting the semantic of the inclusion and extension relations defined in UML [20] for use cases.

Acknowledgements This research has been supported by the Tempros project (TIN2010-20057-C03-02) and Red CaSA (TIN 2010-12312-E) of the Ministerio de Ciencia e Innovación, Spain, and NDTQ-Framework project of the Junta de Andalucía, Spain (TIC-5789).

References

1. Schmidt DC (2006) Guest editor's introduction: model-driven engineering. Computer 39(2):25–31
2. Escalona MJ, Aragón G (2008) NDT. A model-driven approach for web requirements. IEEE Trans Software Eng 34(3):370–390
3. Myers G (2004) The art of software testing, 2nd edn. Addison-Wesley, Boston, MA
4. Denger C, Medina M (2003) Test case derived from requirement specifications. Fraunhofer IESE Report, Germany
5. Escalona MJ, Gutiérrez JJ, Mejías M, Aragón G, Ramos I, Torres J, Domínguez FJ (2011) An overview on test generation from functional requirements. J Syst Software: Elsevier 84(8):1379–1393
6. Fröhlich P, Link J (2000) Automated test case generation from dynamic models. ECOOP 2000, pp 472–491
7. Naresh A (2002) Testing from use cases using path analysis technique. In: International conference on software testing analysis & review
8. Mogyorodi GE (2002). What is requirements-based testing? In: 15th Annual software technology conference, Salt Lake City, USA, 28 Apr to 1 May
9. Boddu R, Guo L, Mukhopadhyay S (2004) RETNA: From requirements to testing in natural way. In: Proceedings of 12th IEEE international requirements engineering RE'04
10. Ruder A (2004) UML-based test generation and execution. Rückblick Meeting. Berlin
11. Binder RV (1999) Testing object-oriented systems. Addison Wesley, Boston, MA
12. Ibrahim R, Saringat MZ, Ibrahim N, Ismail N (2007) An automatic tool for generating test cases from the system's requirements. In: 7th International conference on computer and information technology, Fukushima, Japan
13. Ostrand TJ, Balcer MJ (1988) Category-partition method. Communications of the ACM, pp 676–686
14. Gutiérrez JJ, Nebut C, Escalona MJ, Mejías M, Ramos I (2008) Visualization of use cases through automatically generated activity diagrams. Lect Notes Comput Sci 5301:83–96
15. Supporting web www.iwt2.org/mdetest. Last updated 15 Apr 2012
16. Kapova L, Goldschmidt T, Becker S, Henss J (2010). Evaluating maintainability with code metrics for model-to-model transformations. Research into practice–reality and gaps, Springer, pp 151–166
17. Object Management Group (2011) Query View Transformation Specification 2.0. http://www.omg.org. Accessed 7 Jan 2012
18. García-García J, Cutilla CR, Escalona MJ, Alba M, Torres J (2011) NDT-Driver, a Java Tool to Support QVT Transformations for NDT. In: 20th International conference on information systems development, Edinburgh, Scotland, 24–26 August
19. Cutilla CR, García-García JA, Alba M, Escalona MJ, Rodríguez-Catalán L (2011) Aplicación del paradigma MDE para la generación de pruebas funcionales. Experiencia dentro del proyecto AQUA-WS. In: Proceeding of Automating Test Case Design, Selection and Evaluation ATSE 2011, Chaves, Portugal
20. Object Management Group (2011) Unified Modeling Language 2.4. www.omg.org. Accessed 24 June 2012