An approach for Model-Driven test generation

J.J. Gutierrez, M.J. Escalona, M. Mejías, I. Ramos, J. Torres
Department of Computer Languages and System
University of Seville
Seville, Spain
{ mjescalona, javierj, risoto, isabel.ramos, jtorres}@us.es

Abstract—**The test phase is one of the most important phases in software development. However, in practice, little research has been carried out in this field. Model-Driven Engineering is a new paradigm that can help to minimize test cases generation costs and can ensure quality of results. This paper presents the application of the MDE paradigm in the systematic, even automatic, generation of System Test Software.**

Keywords-component; Model-Driven Engineering, Test generation

I. INTRODUCTION

The Software development process is composed of a set of phases and tasks that the development team must carry out according to specific scheduling and requirements.

Two of the most costly phases in the life cycle of a software system are the test phase and the maintenance phase. In both phases, it must be verified that the developed system covers the user's requirements [21].

However, development delays or cost problems frequently provokes that the test phase is insufficient or even not executed, and that the tests are considered unsuitable in the maintenance phase.

One possible solution to this common problem is to make the test execution easier for the development team. The definition of test cases and the assurance that they cover user's requirements could help towards test phase generation to this end.

A tendency has asserted in the research community around the definition of systematic approaches for the generation of test cases from functional requirements.

These solutions ensure that the tests cover user's requirements and offer easier ways to generate test cases.

In this environment, the use of the Model-Driven paradigm not only offers suitable results but test cases can also be generated from functional requirements.

This paper presents an approach based on Model-Driven Engineering (MDE)[11] for the systematic generation of system test cases from functional requirements.

The paper is divided in five sections. In Section II, a short survey about the current situation of systematic test generation from functional requirements is presented, together with a short introduction of the MDE paradigm.

In Section III, a global vision is given of the MDE test generation approach. A brief overview of its metamodels, transformations, specific syntax and associated tools are presented.

In Section IV the approach is illustrated with a simple example. Starting with the use cases, a complete application of the approach is presented.

In the conclusions, this paper introduces references to real practical applications and future research lines using this approach.

II. CURRENT SITUATION

A. *Systematic Test Generation*

The test phase is one of the most important phases in the software development process. However, delays in development may be caused by incorrect execution. For this reason, several research teams are working on test cases generated directly from requirements. These groups work on offering effective processes for the systematic generation of test products in order to consume the shortest time possible and to cover a high number of tests.

Although an extensive list of references may be found in [5] and [12], the following paragraphs describe some of the most relevant approaches along these lines.

Binder [3] describes the Extended Use Case Test pattern, based on the Category-Partition method. This pattern is focused on the identification of operational variables and builds up a decision table with all the combinations between values and the expected results for each set of values. The main ideas of Section 3 have been extracted from this pattern; however, it is less formal than our approach and does not support any automation nor generate test scripts.

The TDE/UML approach, [23], expresses a use case as a Unified Modelling Language (UML) activity diagram and uses the Category-Partition method [20] to generate test cases. The diagram is annotated with variables, categories, partitions and conditions. A proprietary test tool calculates all the possible combinations between the paths and categories that fit all the conditions. However, the approach does not indicate if the activity diagram may be generated automatically from the use cases, nor the format in which the use cases must be

defined. The TDE/UML approach might generate executable test scripts, but no expected results and validation actions are generated.

TOTEM [15], Requirement-based Contract [16] and the CowSuite [1] approaches express a use case as a UML sequence diagram. The sequences of messages are expressed as regular expressions and are combined to generate test cases. We found some problems using sequence diagrams: It is very difficult to express alternative or erroneous sequences in the same diagram. Information about architecture and internal implementation, such as classes and messages are also needed, so sequence diagrams cannot be applied in early development phases.

The RETNA approach [4] uses a paragraph of non-format text to define the use case and it applies language processing techniques to extract information, build state machines and generate test cases. This approach introduces strong restrictions in use cases, and needs a wide range of tools and techniques, mainly of language processing, which makes it difficult to apply.

Other approaches work directly with natural language, such as those in [14] and [16]. They all propose a simple combinational explosion among all scenarios in a use case. These approaches are quite simple and omit many important aspects, such as coverage, test values, expected results and test implementation.

B. Model-Driven Engineering and Model-Driven Architecture

MDE is a new paradigm for software generation where concepts have the greatest importance, independent of their representation. MDE proposes representing concepts using metamodels. The development process is supported by a set of transformations and relations between concepts that leads to agile developments and ensures consistence between models. Metamodels offer an abstract artifact representation for any environment. Thus, MDE presents any abstract concept and enables different representations for each concept that can be selected by the development team.

On the other hand, MDA (Model-Driven Architecture) is the standard Model-Driven Architecture defined by OMG[18]. It is oriented towards the definition of a common structure and a common language to define MDE approaches. MDA proposes four levels:

- CIM (Computer Independent Model): In this level, concepts that capture the logic of the company are defined. For instance, the business or the requirements models are included in this level.
- PIM (Platform Independent Model): This level involves the concepts that define the software system if any refer to the specific development platform. For instance, analysis artifacts are included in this level.
- PSM (Platform Specific Model): Here, models that depend on the specific development platform are defined, for instance, specific models for Java or .Net.
- Code: This is the highest level and includes the implementation of the system.

In MDA between these levels some transformations can be defined: CIM-to-PIM, PIM-to-PSM or PSM-to-code transformations can be defined. However, transformations in the same level, for instance PIM-to-PIM, can be also defined in MDA.

MDE offers a new way of building software. Since results and models are obtained from previous models using transformations, MDE ensures the traceability of products. Thus, by using CIM-to-PIM transformations for instance, an analysis model could be obtained from the requirements model of a system.

Furthermore, MDE frequently reduces the development time. Transformations offer a systematical way to derive products. However, if a suitable tool is used, these transformations can be automatically executed and the development team can be reduced as presented in [7].

Nowadays, MDE is being used in different contexts: Web Engineering, Aspect Software Programming, Service Oriented Development, etc. This paper is focused on analyzing how MDE can improve the generation of test systems directly from requirements.

III. AN APPROACH FOR TEST GENERATION

In this Section, an approach based on MDE for test system generation is introduced.

This approach starts with the definition of functional requirements and proposes a systematic process, based on QVT transformations, to derive system tests. Figure 1 introduces the context of the approach presented in the MDA environment. Although in Figure 1 the whole process is presented, our approach is only focused on the CIM and PIM levels only. In fact, the MDE test generation approach starts with functional requirements defined in the CIM level. With a set of transformation CIM-to-PIM a set of models in the PIM level are obtained.

Specifically, a set of Test Scenarios are obtained using QVT transformations. A test scenario is a concrete execution path in the system that it is derived from functional requirements.

Therefore, with CIM-to-PIM transformations a set of operational variables are also derived. An operational variable is any element which value may change among different executions of a functional requirement.

With both sets of artefacts (execution paths and operational variables), which are related in the approach, and using some PIM-to-PIM transformations, the systematic derivation of system test cases is enabled.

In general, this is the approach. Starting from the functional requirements definition, a set of system test cases can be derived. However, several aspects have to be concreted in this general introduction. In next sections, metamodels, transformations, tools and the concrete syntax used in this approach are defined.

The process must continue with the PSM generation. From the test cases definition, some executable test cases must be design. Thus, with a set of PIM-to-PSM transformations,

test cases definition must be adapted to a concrete platform like .NET, J2EE, etc. From these PSM definitions, the code with the executable test generation may be obtained.

A. *Metamodels*

In order to offer suitable environments for the definition of models presented in Figure 1, a set of metamodels must be included.

UML class diagram is one of the most used notations for the definition of metamodels. In our approach, two metamodels has been defined:

1- The metamodel for functional requirements
2- The metamodel for functional test

In the first metamodel, artifacts related with functional requirements are presented. It is very important to stick out that any representation reference is included in this metamodel.

The metamodel, which is presented in Figure 2 and is studied completely in [13], only included the concepts. Thus, for each functional requirement (represented by the *FunctionalRequirement* metaclass) is composed of a set of Main Steps (the *MainStep* metaclass). Some of these main steps can produce some exceptional ways in the execution (the *ExceptionalStep* metaclass), for instance, error paths or alternative routes.

Besides, the metamodel defines that every main step must be executed by an Actor (represented by the *FRActor* metaclass).

The metamodel for functional testing is a metamodel that grouped every artefact in the PIM level. Thus, this metamodel is composed by a set of metaclasses that grouped the abstract definition of test scenarios, operational variables and system test cases.

This metamodel could be defined as a set of three metamodels that were grouped in the PIM level.

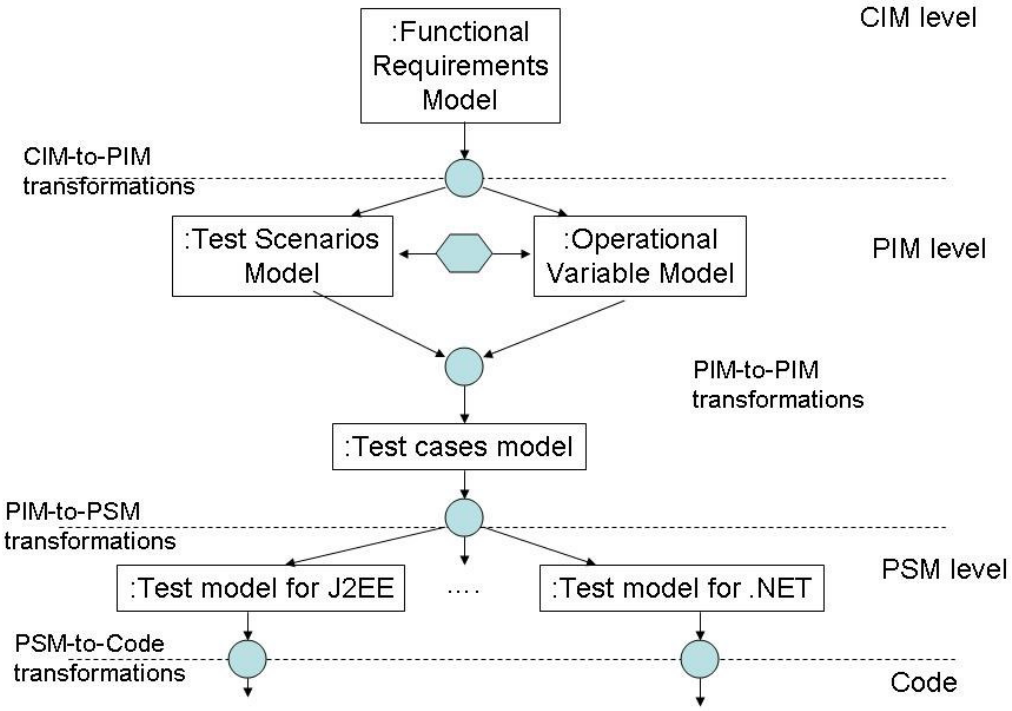

Figure 1. MDE Test Generation Approach

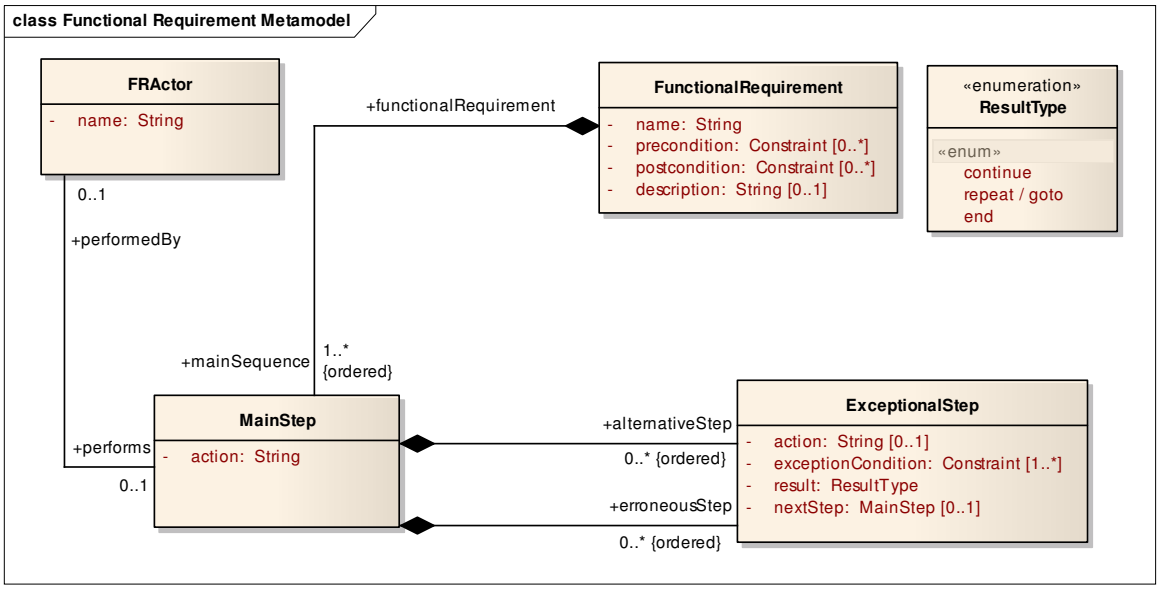

Figure 2. Functional Requirements Metamodel

Some constraints could be defined among the concepts defined in each metamodel. These constraints, defined in each class as invariants, limit the use of concepts and concretize their scopes. For example, in the metamodel for functional test there is a metaclass named *TestStep*. This metaclass represents a step that must be executed as part of one test case. It is related with two metaclasses in the metamodel: *SourceExceptionalStep* and *SourceMainStep*. These metaclasses are derived from *MainStep* and *ExceptionalStep* in the Functional Requirements Metamodel (see Figure 2). Both metaclasses express that any step in a test case must be derived from an initial step in the functional model. Therefore, an OCL constraint can be defined involving the *TestStep*, *SourceMainStep* and *SourceExceptionalStep* metaclasses. This constraint is introduced in Table I.

TABLE I. AN OCL CONSTRAINT

```
self.SourceExceptionalStep → isEmpty()
     implies not(self.SourceMainStep)
                  → isEmpty()) or
self. SourceMainStep  → isEmpty()
     impliesnot(self.MainExceptionalStep
                  → isEmpty())
```

Basically, the constraint from table 1 expresses that any *TestStep* element must be originality derived from a *SourceExceptionalStep* or from a *SourceMainStep*.

B. *Transformations*

Any MDE approach is composed of two parts: metamodels and transformations. Transformations are the way to "produce" results in a MDE approach.

In the MDE Test approach, several transformations were defined.

There are several ways to define transformations in the MDE environment. In this case, QVT was the selected language [22].

Two different ways can be followed when using QVT: Relational QVT and Operational QVT. The first one is easier to be understood, however, some specific aspect are very difficult to be expressed with Relational QVT. For this reason, in the test generation approach a mix selection was used. Transformations are commonly defined with Relational QVT and, only in those cases that it is difficult or even impossible the use of Relational QVT, Operational QVT is used.

C. *Concrete Syntax*

In practice, metamodels and transformations are not useful enough. They are the abstract definition of a set of concepts that must be enriched with a concrete syntax to represent these concepts.

The decision of a concrete syntax is not easy. We define two different possibilities in our approach.

A. ***UML profiles***. A profile is a useful tool defined by UML[19] that enables the definition of concepts as formal extensions of UML artifacts. For each metamodel defined in our approach, a concrete profile was defined. The elements of each metamodel (such as *FunctionalRequirement* or

TestStep metaclasses) has been modeled by extending existing UML metaclases like *Class* and *Operation*.

A profile adds important advantages in MDE approaches.

1. It offers a standard vision of the approach. If anybody wants to use our approach, they can understand concepts in based of the standard definition of UML. Thus, if *MainStep* (see Figure 2) is defined as a formal extension of an UML Activity, anyone who knows UML can easily understand its meaning.
2. It offers a suitable way for the use of tools. Most of commercial or UML tools (such as Enterprise Architecture[10], Rational Rose [23], StartUML [25], etc.) include the possibility of defining profiles. Thus, if this possibility is used and a concrete profile is defined, for example in Enterprise Architect, the development team automatically can define our approach artifacts in Enterprise Architect. This possibility was used in practice with our approach. It is presented in posterior sections.
3. It increases the compatibility with other approaches. As we analyzed in Figure 1, our approach only covers CIM and PIM level. The use of standards makes easier the compatibility with other approaches that works in PSM or Code levels. In [8] some important references about this advantage can be found.

B. ***The definition of concrete syntax***. Apart of the definition of profiles, our approach also included some specific recommendation for the concrete syntax that may be used in each metamodel. In fact, these syntaxes are going to be introduced in section IV with the basic example. Thus, for the functional requirements metamodel, the approach stakes by language patterns and activity diagrams.

Language patterns are a textual way to express requirements. Concretely, the approach uses NDT (Navigational Development Techniques)[9] functional patterns. However, in order to automate the treatment of these patterns some normalization must be done. In order to analyze each path in functional requirements, the use of activity diagrams is simpler. They offer a graphical notation that makes easier to find each path in the functional requirements definition. Patterns are easier to be completed with users, because they use the own vocabulary of the users. But activity diagrams are clearer for test generation.

For the metamodel of functional test, different notations are used. For test scenarios and operational variables some specific textual patterns were defined. We proposed to use XML form defining these artefacts.

In section III, a complete example with this notation is presented.

D. Tools

According of some comparatives developed in previous works [12], there is an important lack of tools to support system test generation. For this reason, as part of this project, an executable proof-of-concept tool has been developed. This tool is based in Java and it uses a XML representation of the functional requirement as input and generates a XMI activity diagram representation of the functional requirements, a set of paths through the activity diagram a set of operational variables and partitions [3] and a script that automatically calculate all valid combinations among partitions.

This proof-of-concept tool has generated an invaluable amount of information used in the formalization and the practical testing of the transformation approach. After that, the tool was improved with support for use cases generated by Sparx Enterprise Architect[10] and with XMI tailored for StarUML[25] and Enterprise Architect tools.

This approach has also enterprise experience in the use of supporting tools. As it was introduced, this approach uses NDT functional requirements definition. NDT is a Model-Driven Web approach that uses this paradigm to generate analysis models from requirements models systematically. NDT has an associated group of tools, NDT-Suite. NDT-Suite is composed by four tools:

1- **NDT-Profile**: This is a specific profile for NDT, developed using Enterprise Architect. This tool offers an environment to define specific profiles and NDT-Profile has adapted Enterprise Architect to support each artifact of NDT.

2- **NDT-Driver**: This is a tool to execute transformations of NDT. NDT-Driver is a Java-free tool which implements QVT Transformations in NDT and allows analysis models to be obtained automatically from the requirements model.

3- **NDT-Quality**: This is a tool that checks the quality of a project developed with NDT-Profile.

4- **NDT-Report**: This is a tool that prepares formal documents in order to be validated by final users and clients. For instance, it enables the automatic generation of a Requirements Document with the format defined by clients.

All these tools and their manuals can be downloaded from www.iwt2.org.

The presented test generation approach was recently included in this group of tools and, nowadays, it is being used in some projects in the Ministry of Culture and the local water company (EMASESA)[6].

The easier way to include the test generation approach in this tool environment is based in the powerful use of metamodels and profiles. Enterprise Architect, as other important commercial tools which supports UML, offers simple ways for profile definitions.

Thus, in this case, the approach profiles were included in NDT-Profile and easily both approaches were integrated.

IV. A BASIC EXAMPLE

In order to illustrate the presented approach, a simple example is presented in this section. The starting point of the approach is the definition of functional requirements as a use case diagram. In Figure 3 a simple use case diagram of a web application to manage an online link catalogue is presented[1].

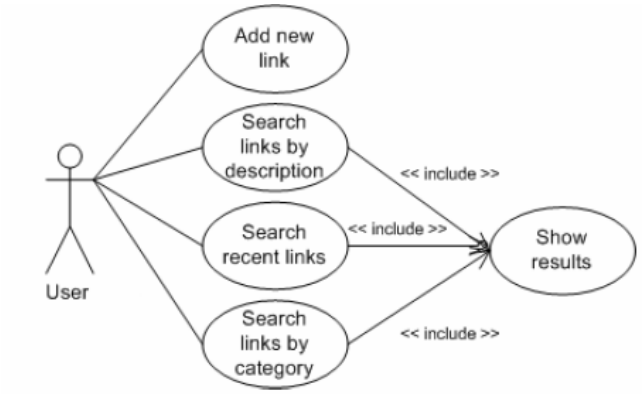

Figure 3. An use case diagram for the example

A user can add a new link or search links in three different ways: by their descriptions, by their categories or see only the most recent links.

Each of these use cases must be described using a pattern. A pattern is a table with specific fields that must be defined with users. In Table II, the pattern for Search links by description use case is presented. As can be observed, this pattern included any concept, relation and attributed presented in the metamodel in Figure 2. This pattern offers the first concrete syntax for our metamodel.

Each path in this pattern is offering a suitable test way when the system will be developed. However, no all paths can be defined as a test case because it is impossible to cover the complete number of tests.

With the textual notation presented in Table II, it is quite difficult to analyze each of these execution paths. Thus, using transformations defined in [13], an activity diagram can be automatically defined.

This activity diagram is presented in figure 4. In this case, any possible execution paths are easier to be analyzed.

After generating the activity diagram, a set of paths are discovered. Each path is a test scenario and a potential test case. In table III, derived paths are obtained. For get results expressed in table III some CIM-to-PIM transformations were executed (see Figure 1).

The second set of CIM-to-PIM transformations lets obtain operational variables. Operational variables can be identified in actions which express some entries or exits in the system or in decision nodes in the execution route. These are decision nodes in the activity diagram (Figure 4). For each operational variable its domain must be also identified. These can be identified in each decision node in the execution route (again decision nodes in the activity diagram). Each path in the decision node is a possible value for the operational variable. Thus, in the example, three operational variables can be obtained (one for each decision node):

- Variable D1: Is the search cancel?
  - Possible values: Yes or not
- Variable D2: Is the description empty?
  - Possible values: Yes or not
- Variable D3: Are there errors or empty results?
  - Possible values: Errors, empty results or OK

Combining possible values of test scenarios models and operational values the test cases models can be obtained. A possible syntax to express test cases is presented in table IV. In this case, the test case is obtained from the first test scenario, as can be observed in table IV.

In this case, a concrete pattern was used to define this test case. *Name* and *description* are some fields to identify this test case. *Source field* stores the use case that produces this test case. *Initial state* and *final state* indicates the starting and the final point in the test case execution. They must be completed when the test case is tested. *Test information* and *final results* fields store initial values and final values of some external variables and information for the test (data used in the test execution). *Priority* expresses which the priority of the test is and *comments* field stores some relevant comments for the test case.

Actions field describes steps that must be followed in the test case. Actions are derived automatically with the MDE approach.

Other aspects are not obtained automatically and they must be completed by the development team. There are some studies to analyse for instance, the priority automatically, based on the probability of the execution paths. Even the automatic generation of data to test the use case could be possible, although this aspect is not yet included in our approach.

[1] This example is available in www.codecharge.com

TABLE II. A PATTERN EXAMPLE

Name	UC-02. Search link by description
Preconditions	NO
Main Sequence	1. The user asks the system for searching links by description. 2. The system asks for the description. 3. The user introduces the description. 4. The system searches for the links which matches up with the description introduced by the user. 5. The system shows the results.
Errors/alternatives	3.1.1. At any time, the user may cancel the search, and then the use case ends. 4.1.p. If the actor introduces an empty description, then the system searches for all stored links and the result is to continue the execution of this use case.4.2.i. If the system finds any error performing the search, then an error message is shown and this use case ends. 5.1.i. If the result is empty, then the system shows a message and this use case ends.
Results	1. The system shows the results of UC-05 3.1.i. Out of the limits of this use case. 4.2.i. Error message. 5.1.p. Message of no found results
Post condition	NO

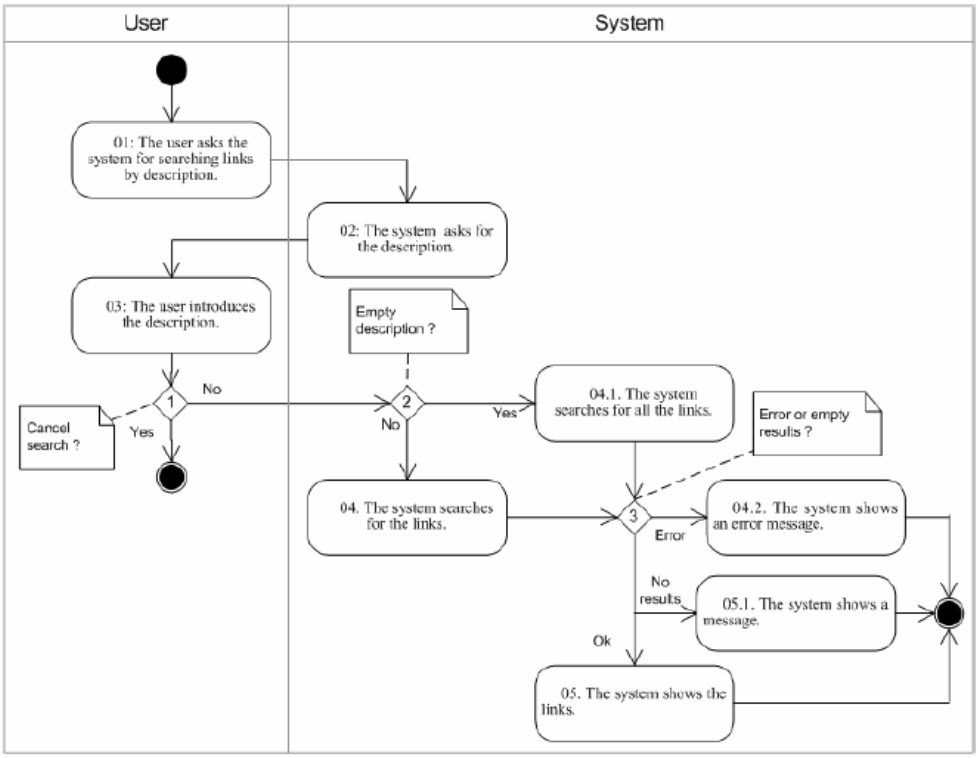

Figure 4. The activity diagram

TABLE III. TEST SCENARIOS

Id	Path
1	01 -> 02 -> 03 -> D1(No) -> D2(No)-> 04 -> D3(No error & Results) -> 05
2	01 -> 02 -> 03 -> D1(No) -> D2(No)-> 04 -> D3(No error & No Results) -> 05.1
3	01 -> 02 -> 03 -> D1(No) -> D2(No)-> 04 -> D3(Error) -> 04.2
4	01 -> 02 -> 03 -> D1(No) -> D2(Yes)-> 04.1 -> D3(No error & Results) -> 05
5	01 -> 02 -> 03 -> D1(No) -> D2(Yes)-> 04.1 -> D3(No error & No Results) -> 05.1
6	01 -> 02 -> 03 -> D1(No) -> D2(Yes)-> 04.1 -> D3(Error) -> 04.2
7	01 -> 02 -> 03 -> D1(Yes)

TABLE IV. TEST CASE

Name	TC-01		
Description	-		
Source	UC-02. Search link by description		
Initial State	-		
Final results	-		
Test information	-		
Actions	**Index**	**Body**	**Test data**
	1	The user asks the system for searching links by description	
	2	The system asks for the description.	
	3	The system asks for the description.	D1 = No D2= No
	4	The system searches for the links which matches up with the description introduced by the user	
	5	The system shows the results.	D5 = No errors
Final states	-		
Priority	-		
Comments	-		

Test case definition using pattern like table IV is only one possible way to represent it. Nowadays, we have a tool that generates this test case definition in a XML file that is, probably easier to be implemented in a PSM level. However, patterns offer a more suitable way to understand the test definition.

V. CONCLUSIONS

This paper has introduced an approach based on MDE paradigm for the systematic generation of system test cases from functional requirements.

The paper starts with a presentation of the actual situation and a short introduction about Model-Driven paradigm.

After that, a global description of the approach and an example are introduced.

This proposal needs to define one CIM model (Functional Requirements model) and three PIM models (Test Scenarios, Operational Variable and Test Cases models) corresponding to the MDA paradigm. Also, we have included the transformations between models. These transformations can be defined by QVT. In this case a mix solution has been adopted and we have used Relational QVT and Operational QVT to define different aspects of the transformations.

As it was presented in Section III, this approach is being used and integrated in the enterprise environment. Thus, at the end of 2007, the test case metamodel was

integrated into NDT-Profile and it was used in a complex project named Mosaico[7].

This project is developed in collaboration with the Ministry of Culture in Andalusia and it manages information about historical heritage.

In fact, in this organization, NDT is applied in all its software development projects. And, nowadays, the test generation approach is also being included in these software projects.

The feedback obtained from the enterprise environment is being an important improvement source. This is one of our future research lines. In fact, from the first application, in Mosaico, important comments and suggestions were obtained for the improvement of the approach.

Another important detected aspect is to continue with the complete process. As it was presented in Figure 1, our approach only covers the CIM and the PIM level. The approach considers CIM-to-PIM transformations and PIM-to-PIM. The complete definition of the approach, including PIM-to-PSM and PSM-to-Code and metamodels in this level is a very interesting fact.

Other important open line is the definition of objective metrics which can measure the quality of the result in the approach.

Obviously, tools are always interesting. Although the approach was integrated in NDT-Suite, the research in new tools possibilities is always interesting.

ACKNOWLEDGMENT

This research has been supported by the project QSimTest (TIN2007-67843-C06_03) and by the RePRIS project of the Ministerio de Educación y Ciencia (TIN2007-30391-E), Spain.

We would like to thank Antonio Molina González, Emilio Martinez Force and Antonio Gómez Rodríguez by their interesting practical feedback in the real application of this approach. Mainly in the Mosaico project.

REFERENCES

[1] ArgoUWE. http://www.pst.informatik.uni-muenchen.de/projekte/argouwe

[2] F. Basanieri, A. Bertolino, E. Marchetti. The Cow_Suite Approach to Planning and Deriving Test Suites in UML Projects. Lecture Notes In Computer Science 2460 pp. 383-397. 2002.

[3] R. V. Binder. Testing Object-Oriented Systems. Addison-Wesley. USA. 2000.

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

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

[6] Emasesa. Empresa Municipal de Aguas de Sevilla. http://www.aguasdesevilla.com

[7] M.J. Escalona, Equipo de Coordinación. MOSAICO. El Sistema de Informacion para la gestión del Patrimonio Historico Andaluz Proceedings of XI International Congress on Project Engineering ISSN: 978-84-690-8134-1. Spain, 2007.

[8] M.J. Escalona, N. Koch- Metamodelling the Requirements of Web Systems. Lecture Notes in Bussiness Information Process. Web Information Systems and Technologies: Int. Conferences WEBIST 2005 and WEBIST 2006. Springer Verlag Vol.1, pp-.267-288 ISSN: 1865-1348. USA. 2007

[9] M.J. Escalona, G. Aragón. NDT. A Model-Driven approach for Web requirements. IEEE Transaction on Software Engineering. Vol. 34. Nº3. pp.370-390. 2008.

[10] Enterprise Architect. www.sparxsystems.com

[11] F. Fondement and R. Silaghi.Defining Model Driven Engineering Processes. 3rd Workshop in Software Model Engineering (WiSME 2004)Lisbon, Portugal

[12] J.J. Gutiérrez, M.J. Escalona, M. Mejías, J. Torres. Comparative Analysis of Methodological Proposes to Systematic Generation of System Test Cases. 3º Workshop on System Testing and Validation. Paris. France. 2004.

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

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

[15] Y. Labiche, L.C. Briand. A UML-Based Approach to System Testing, Journal of Software and Systems Modelling (SoSyM) Vol. 1 No.1 pp. 10-42. 2002.

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

[17] C. Nebut, F. Fleury, Y. Le Traon, J.M. Jézéquel. Automatic Test Generation: A Use Case Driven Approach. IEEE Transactions on Software Engineering Vol. 32. 3. March. 2006.

[18] OMG: MDA Guide, http://www.omg.org/docs/omg/03-06-01.pdf. Version 1.0.1. 2003.

[19] OMG. Unified Modeling Language: Superstructure, version 2.0. Specification, OMG, 2005. http://www.omg.org/cgi-bin/doc?formal/05-07-04.

[20] T.J. Ostrand, M. J., Balcer. Category-Partition Method. Communications of the ACM. 676-686. 1998.

[21] R.S. Pressman. Software Engineering. A practitioner's Approach. Sixth Edition.McGraw Hill, 2005

[22] 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.

[23] http://www-01.ibm.com/software/awdtools/developer/rose/

[24] A. Ruder. UML-based Test Generation and Execution. Rückblick Meeting. Berlin. Germany. 2004.

[25] StarUML.http://staruml.sourceforge.net/en/