NDT-Driver: A Java Tool to Support QVT Transformations for NDT

J.A. García-García, C.R. Cutilla, M.J. Escalona, M. Alba, and J. Torres

1 Introduction

The model-driven engineering paradigm (MDE) came up in order to tackle the complexity of platforms and the inability of third-generation languages to relieve this complexity and effectively express the domain concepts of the problem. This new paradigm, apart from raising the level of abstraction, intends to increase automation during the life cycle of software development.

MDE works, as the primary form of expression, with definitions of models and transformation rules among these models entailing the production of other models. One of the languages to describe the transformation rules is QVT (OMG 2008) or query/view/transformation language. QVT standard defines a declarative and imperative language proposed by the OMG (Object Management Group) for model transformation in the context of MDE.

However, QVT notations are not easy to be applied in practical environments because it does not result too friendly for development teams. Concepts such as models, metamodels, transformations, or QVT are not common notations in the enterprise environment, and they seem too abstract and complex. For this reason, this research paper presents how the NDT methodology, acronym for Navigational Development Techniques, addresses this challenge with the aim of involving the enterprise with the power of the model-driven paradigm.

The chapter is structured as follows. After this introduction, Sect. 2 briefly studies how to transform some methodologies that belong to model-driven engi-neering paradigm. Section 3 provides an overview of NDT methodology and their tools. Section 4 presents NDT-Driver, a tool that implements a set of automated

J.A. García-García (✉) • C.R. Cutilla • M.J. Escalona • M. Alba • J. Torres
IWT2 Group, University of Seville, Seville, Spain
e-mail: julian.garcia@iwt2.org; carmen.ruiz@iwt2.org; mjescalona@us.es; manuel.alba@iwt2.org; jtorres@us.es

procedures to generate different models in NDT life cycle. Finally, in Sect. 5, final conclusions and lessons learned are detailed, and the chapter concludes with future work in Sect. 6.

2 Related Works

Currently, there is a wide range of Web methodologies that belong to the MDE paradigm and define its procedure and transformations from one model to other model. For this reason, only the most referenced ones will be briefly described:

- OOHDM (Rossi and Schwabe 2008) is a highly referenced methodology and one of the most accepted. It consists of four different design activities: conceptual model, navigational design, abstract interface design, and implementation. The conceptual and navigational transformations are completely independent, except from operation invocations of conceptual PSM objects from the navigational PSM, where the type of invocation may vary. Thus, the implementation technology and platform of the conceptual PSM and the navigational PSM may be selected and combine quite independently.
- UWE (Koch et al. 2008) is a methodology based on Unified Modeling Language (UML 2005) for Web application development. It covers the whole development life cycle of Web systems from the requirements specifications to code generation, focusing on personalized or adaptive applications. The process of development is based on three main phases: the phase of capture of requirements, the phase of analysis and design, and the phase of implementation.
  The transformation rules are defined to map metamodel WebRE to UWE and UWE to metamodels. The first model transformation step of the UWE process consists of mapping Web requirements models to UWE functional models. The design models are content, navigation, process, presentation, and adaptation. There is a set of dependencies among these functional models themselves that allow the creation of other models, as well as their amendments.
  ArgoUWE and MagicUWE have been developed for the computer-aided design of Web applications using UWE.
- WebML (Ceri et al. 2000) is a proposed methodology based on formal specification graphics into a complete design process. It is used on applications that highly interact with information.
  The model is divided into four phases: the process definition phase, which uses the notation BPMN; the data model phase, which is defined by different data tables and their relationships and uses graphs of entity-relation or UML class diagrams; the hypertext model phase, which consists in the composition model, representing a hypertext sites where every site has content elements, and the navigation model, which defines the links among different sites; and finally, the composition model, which details content items of each site.

WebML is associated with a development tool called WebRatio, which automatically generates fully functional applications from WebML diagrams.

- OOWS (Fons et al. 2003) is an extension of OO-Method. The process to define a software system for a Web environment is based on two steps: the conceptual model and the navigational model. The conceptual model is the specification of user requirements. The navigation model is based on an object model and navigation requirements using UML notation. The navigation model is composed of a set of navigation maps (one for each agent) to represent and structure the global vision system by defining allowable navigation. This is directly represented by means of a directed graph where nodes are navigational contexts and arches represent the navigation links.

  Once the navigational model is defined, the presentation characteristics are associated with the system. Presentation requirements will be based on the use of simple presentation patterns related to the different elements that conform a navigation node. The generator (compiler) will use this information stored to create the diverse interfaces for each user within the architecture of Web application that the OOWS method proposes.

3 An Overview of NDT and NDT-Suite

The proposed methodology NDT (Escalona and Aragón 2008), acronym for Navigational Development Techniques, belongs to the MDE paradigm.

Initially, NDT dealt with the definition of a set of formal metamodels for the requirements and analysis phases. In addition, NDT defined a set of derivation rules, stated with the standard QVT, which generated the analysis models from requirements model.

Nowadays, NDT defines a set of metamodels for every phase of the life cycle of software development: the feasibility study phase, the requirements phase, the analysis phase, the design phase, the implementation phase, the testing phase, and finally, the maintenance phase. Besides, it states new transformation rules to systematically generate models. Figure 8.1 shows the first part of the NDT life cycle[1].

The main goal of the requirements phase is to build the catalogue of requirements containing the needs of the system to be developed. It is divided into a series of activities: capture, definition, and validation of requirements.

NDT classifies project requirements according to their nature: information storage requirements, functional requirements, actor requirements, interaction requirements, and nonfunctional requirements. In order to define them, NDT provides special patterns and UML techniques, such as the use case technique for functional requirements specification.

[1] More information about the NDT full life cycle can be found in www.iwt2.org

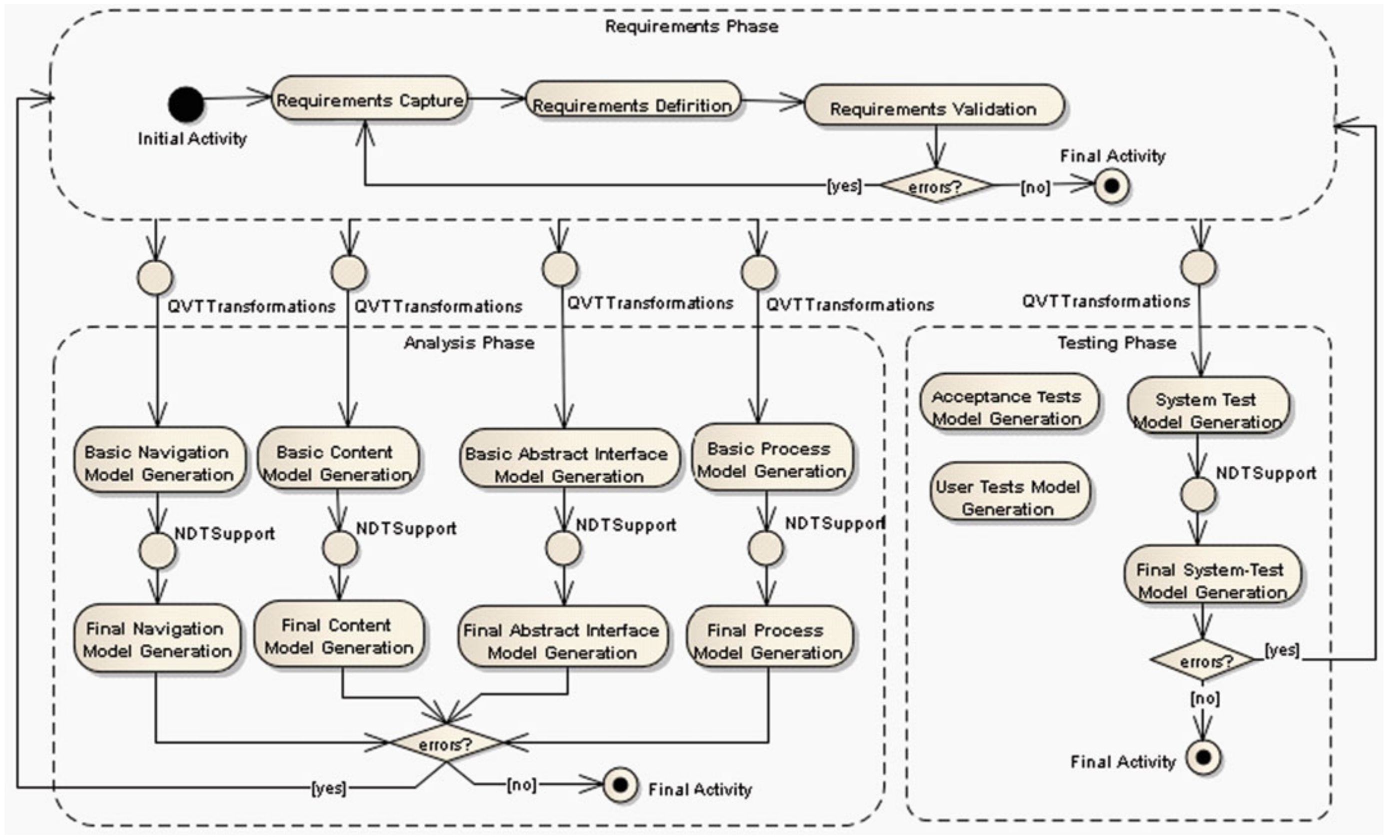

Requirements Phase
Initial Activity
Requirements Capture
Requirements Defirition
Requirements Validation
errors?
[yes]
[no]
Final Activity
QVTTransformations
QVTTransformations
QVTTransformations
QVTTransformations
QVTTransformations
Analysis Phase
Basic Navigation Model Generation
Basic Content Model Generation
Basic Abstract Interface Model Generation
Basic Process Model Generation
NDTSupport
NDTSupport
NDTSupport
NDTSupport
Final Navigation Model Generation
Final Content Model Generation
Final Abstract Interface Model Generation
Final Process Model Generation
errors?
[yes]
[no]
Final Activity
Testing Phase
Acceptance Tests Model Generation
User Tests Model Generation
System Test Model Generation
NDTSupport
Final System-Test Model Generation
errors?
[yes]
[no]
Final Activity

Fig. 8.1 Transformations from requirements to analysis and from requirements to testing

Once the requirements specification phase has been completed and the catalogue of system requirements has been drafted and validated, NDT defines derivation rules to generate the system test model and the analysis phase models. Figure 8.1 shows all these transformations through the stereotype "QVTTransformation."

NDT conceives the testing phase as an early phase of the software life cycle and proposes to carry it out together with the remaining phases. NDT proposes three models in this phase: the implementation tests model, the system tests model, and the acceptance tests model. The system tests model is the only one that can be generated systematically. NDT proposes derivation rules to generate the basic system tests model from the functional requirements defined in the requirements phase. The team of analysts can perform transformations in order to enrich and complete this basic model. Transformations are represented in Fig. 8.1 through the stereotype "NDTSupport."

The analysis phase will include the resulting products from the analysis, definition, and organization of requirements in the previous phase. At this phase, NDT proposes four models: the conceptual model, which represents the static structure of the system; the process model, which represents the functional structure of the system; the navigation model, which shows how users can navigate through the system; and the abstract interface model, a set of prototypes of the system.

The transition between the requirements and the analysis model is standardized and automated, and it is based on QVT transformations, which translate the concepts of requirements metamodels to the first versions of the analysis models. These models are known in NDT as basic models of analysis. For example, the basic conceptual model of analysis is obtained from the storage requirements defined during the requirements phase.

Thereafter, the team of analysts can transform these basic models to enrich and complete the final model of analysis. As this process is not automatic, the expertise of an analyst is required. Transformations are represented in Fig. 8.1 through the stereotype "NDTSupport." To ensure consistency between requirements and analysis models, NDT controls these transformations by means of a set of defined rules and heuristics.

To sum up, NDT offers an environment conducive to the development of Web systems, completely covering the life cycle of software development. NDT has been applied in many practical environments and has succeeded due to the application of transformations among models, which has reduced development time (Escalona and Aragón 2008).

3.1 NDT-Suite

The application of MDE and, particularly, the application of transformations among models may become monotonous and very expensive if there are no software tools automating the process. To meet this need, NDT has defined a set of supporting tools called NDT-Suite. Currently, the suite of NDT comprises the following free Java tools:

- NDT-Profile is a specific profile for NDT developed using Enterprise Architect (EA 2010). NDT-Profile offers the chance of gathering all the artifacts that define NDT easily and quickly, as they are integrated within the tool Enterprise Architect.
- NDT-Quality is a tool that automates most of the methodological review of a project developed with NDT-Profile. It checks both the quality of NDT methodology in each phase of software life cycle and the quality of traceability of the MDE rules of NDT.
- NDT-Driver is presented in Sect. 4 of this chapter.
- NDT-Prototype is a tool designed to automatically generate a set of XHTML prototypes from the navigation models of a project, described in the analysis phase, developed with NDT-Profile.
- NDT-Glossary implements an automated procedure that generates the first instance of the glossary of terms of a project developed by means of NDT-Profile tool.
- NDT-Checker is the only tool in NDT-Suite that it is not based on the MDE paradigm. This tool includes a set of sheets different for each product of NDT. These sheets give a set of checklists that should be manually reviewed with users in requirements reviews.

4 NDT-Driver

4.1 Overall View

NDT-Driver is one of the main tools of NDT methodology. It is completely based on NDT-Profile.

NDT-Driver is developed in Java and implements a set of automatic procedures for carrying out each of the QVT transformations defined in NDT. It generates the analysis models from requirements, the design models from the analysis, and the tests models from requirements. In addition, NDT-Driver allows obtaining the model requirements from the requirements gathered in the feasibility study phase of the project.

Furthermore, NDT-Driver can be used in projects using both a sequential life cycle and an evolutionary life cycle. Once transformations to perform have been selected, models to generate can be chosen.

To support projects with evolutionary development cycles, NDT-Driver allows the models previously selected to be rebuilt and updated. For example, if all storage requirements have already been defined in the requirements phase, the conceptual model of the analysis phase can be generated. Then, if it is noticed that any storage requirement is not defined according to the user's needs, it is not completely necessary to rebuild the content model because the tool will update it.

NDT-Driver also provides support for projects that develop business systems with service-oriented architecture (SOA).

4.2 QVT Transformations

QVT Transformations offered by NDT are grouped into three categories:

- Requirements to analysis:
  - *Requirements2Content*: this transformation allows the generation of the basic content model from storage requirements.
  - *Requirements2Process*: this transformation allows the generation of the basic model of process classes from the functional requirements.
  - *Requirements2Navigational*: this transformation allows the generation of the basic navigational model from interaction requirements.
  - *Requirements2Prototypes*: this transformation allows the generation of the basic abstract interface model from the requirements definition.
  - *ServicesR2ServicesA*: this transformation allows the generation of the basic service model of analysis phase from the services included in the requirements phase.
- Analysis to design:
  - *Content2DataAccess*: this transformation allows the generation of the basic model of data access classes from the content analysis model.
  - *Content2PhysicalDataModel*: this transformation allows the generation of physical data model from the content model.
  - *Process2Bussines*: this transformation allows the generation of the basic model of business classes from the model of process classes.
  - *Navigational2Presentation*: this transformation allows the generation of the basic model classes presentation from the navigation model.
  - *ServicesA2ServicesD*: this transformation allows the generation of the basic service model of design phase from the services included in the analysis phase.
- Requirements to tests:
  - *Requirements2Test*: this transformation allows the generation of the basic model of system tests from functional requirements.

Table 8.1 *Requirements2Content* transformation (QVT-Java)

QVT transformation

```
transformation Requirements2Content
            (in msr:StorageRequirementsModel, out cm:ContentModel)
{
   main (){
     msr.ObjectsOfType (StorageRequirement)- > map SRtoCL ();
     msr.ObjectsOfType (NewNature)- > map NNtoCLn ();
   }
   mapping StorageRequirement::SRtoCL () : CL {
    name:= 'CL' + self.name ();
    attributes:=self. SpecificField- > map SFtoAttr ();
    links:= self. SpecificField- > map SFtoLinks ();
   }
   mapping SpecificField sf::SFtoAttr (): Attribute
             where {sf.nature- > size () = 0;} {
   name:=self.name ();
    //...
   }
   mapping SpecificField sf, CL cl::SFtoLinks () : Association
               where {sf.nature = conc:Concept{}}{
   name:=self.name ();
   connection = {sf.nature- > map SRtoCL (), cl};   //...
  }  //...
}
```

Java transformation

```
public void requirements2content (StorageRequirementsModel msr){
  Collection Relations col = new ArrayList < Relations > ();
  for (StorageRequirements sr : msr.getByStereotype
    ("StorageRequirement")){
    CL cl = this.createCL (sr);
    for (Attribute a : sr.getAttribute (sr){
       if (a.isBasic ()) this.createAttribute (cl, a);
         else col.add (new Relations (cl, a.getType ()));
    }
  }
  for (NewNature nn : msr.getByStereotype ("NewNature")){/* idem */}
  this.createAssociationLinks (col);//...
}
```

Presenting all these transformations in this chapter is very complex. For this reason, the first one is only presented (see Table 8.1[2]). It states how each storage information requirements has to be translated into a content class in the analysis

[2] Complete information on NDT, their metamodels and transformations can be found in www.iwt2.org

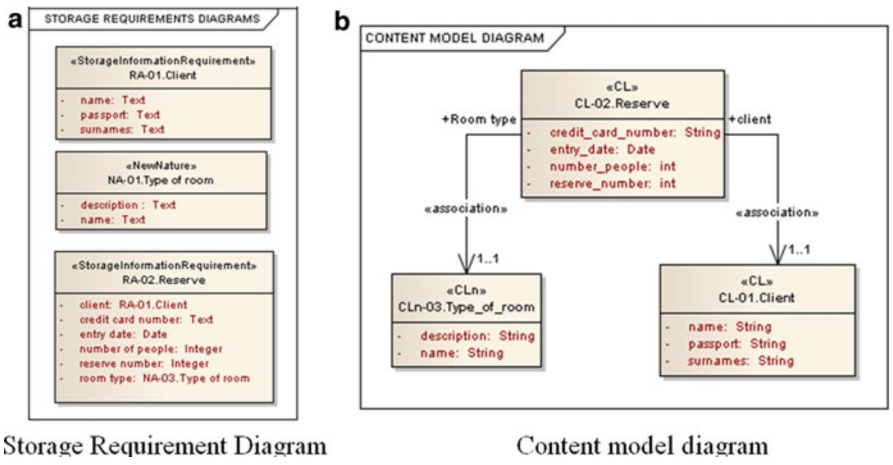


Fig. 8.2 Diagrams of the example (**a**) Storage requirement diagram and (**b**) Content model diagram

phase. Each storage information system generates a class, and each specific field generates an attribute.

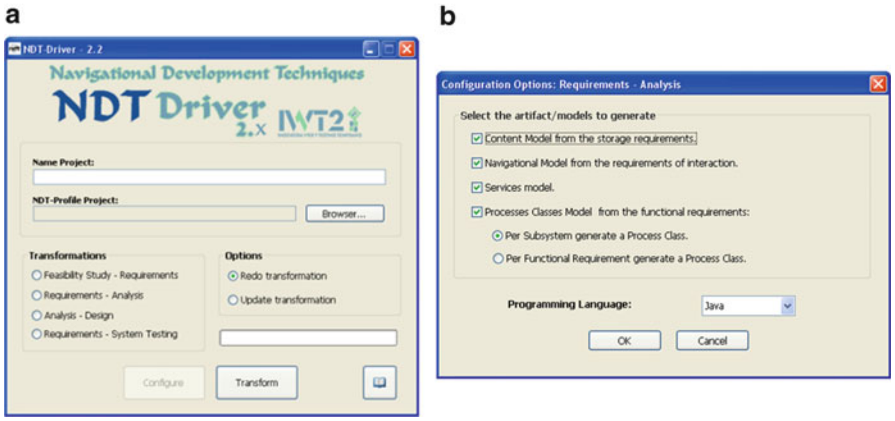
In the *SRtoCL method*, each storage requirement is translated into a class (CL) in the content model. In methods such as *SFtoAttr* and *SFtoLinks*, specific fields are treated. If specific fields have a basic nature, that is, when its relation with content has cardinality 0, they are translated into attributes. If its nature is content, it is translated into association.

4.2.1 A Basic Example

This section provides a representative example of the application of the QVT transformation rule explained in the previous section. The input of this rule is the requirements model.

Figure 8.2a shows the requirements model of a small Web application that manages hotel reservations. In this basic example, important concepts are *reserve*, *client*, and *type of room*. Each concept has its own specific data: name, surname, and passport number of every hotel guest and name and description of each room type. Likewise, each reservation requires the name of the client who books the room, credit card number, date of arrival, number of people per room, and type of room chosen.

After applying the transformation rule shown in Table 8.1, three content classes of analysis will be generated, one for every storage requirement. Figure 8.2b shows a class diagram with the generated content classes, their attributes, and links.


a
b
NDT-Driver - 2.2
Navigational Development Techniques
NDT Driver 2.x
IWT2
Browser...
Configure
Transform
Configuration Options: Requirements - Analysis
OK
Cancel

Name Project:

NDT-Profile Project:

Transformations
( ) Feasibility Study - Requirements
( ) Requirements - Analysis
( ) Analysis - Design
( ) Requirements - System Testing

Options
[✓ Redo transformation]
( ) Update transformation

Select the artifact/models to generate
[✓] Content Model from the storage requirements.
[✓] Navigational Model from the requirements of interaction.
[✓] Services model.
[✓] Processes Classes Model from the functional requirements:
[✓ Per Subsystem generate a Process Class.]
( ) Per Functional Requirement generate a Process Class.

Programming Language: Java

Fig. 8.3 NDT-Driver tool (**a**) Main interface (**b**) Configuration interface

4.3 Interface

The graphical interface of NDT-Driver is very simple and intuitive. Figure 8.3a shows the main interface of the tool. NDT-Driver, as the remaining NDT tools, is available in both English and Spanish. Before carrying out any transformation, some information must be given to the tool.

In the section "Name Project," the name of the project must be entered. This will be useful for the reports generated by the tool. With the "Browser" button, the Enterprise Architect file of the project must be selected. "Transformations" button lets the user select the transformations to be carried out, and "Options" lets the user choose the way the transformation chosen will be carried out.

The "Configure" button selects the models you want to transform. By default, all the models are selected. Figure 8.3b shows the modal window associated with the transformation requirements to analysis. Finally, if clicking on the "Transform" button, NDT-Driver will begin to carry out the transformations selected. At the end of the process, a modal window displays the report of the transformations carried out.

4.4 Practical References

In the last 10 years, NDT and NDT-Suite, and particularly NDT-Driver, were used in a high number of real projects. In fact, NDT-Driver is currently being used in several projects by different companies, either public or privates, big or small. A set of projects developed with NDT was selected, and its tools were used during its life cycle.

4.4.1 AQUA-WS

EMASESA[3] is a company dealing with the general management of the urban water cycle, providing and ensuring water supply to all the citizens in Seville, Spain. AQUA-WS (AQUA-WebServices) project consists in the development and implementation of an integrated business system for customer management, interventions in water distribution and cleaning up, and projects and work management. The development time of this project is estimated in 3 years, and it will finish in 2012. AQUA-WS is very relevant for the use of NDT-Driver in the test phase.

The existing systems are the customer management system (AQUA-SiC), network management system (AQUA-ReD), and the work and projects management system (AQUA-SigO). AQUA-WS arises from the need of unifying all the existing systems into a single one, the core AQUA, by means of the same technology platform.

The project follows an iterative life cycle. Each iteration allows the development team, composed by more than 20 analysts from two companies, to define requirements, studying the previous systems, and introduce them into NDT-Profile. They are checked with NDT-Quality and NDT-Checker and, later, NDT-Driver generates functional test cases. The systematic generation of test cases from functional requirements in the project is providing good support by expediting the validation of their own functional requirements with users. In addition, NDT-Driver can reduce the time analysts spent on specifying system tests and test plans.

4.4.2 Projects for e-Health Systems

NDT was also widely applied in the e-health environment. An example of this is the Diraya project, an information system applied in the Andalusian Health Service, which allows consulting the clinical record of a patient belonging to any hospital center in Andalusia. Its requirements phase was developed by a group of six companies and a high number of analysts. Every company was expert in a specific module of Diraya.

For this project, the development team used the main tools of NDT-Suite. The use of NDT-Profile and NDT-Glossary was essential to guarantee the unification criteria in this multidisciplinary development team. NDT-Quality was also important to assure quality, and NDT-Driver was crucial to reduce the time spent on the project development.

[3] http://www.aguasdesevilla.com

5 Conclusions and Learned Lesson

In this chapter, we present NDT-Driver: a simple and intuitive tool useful for any company which develops a project using NDT methodology in order to take advantage of MDE power, even without knowing this paradigm. Moreover, we describe how the most referenced methodologies perform their transformations, and finally, we briefly introduce NDT and its supporting tools.

As a learned lesson of our experience in the use of MDE in Web treatment, we could conclude that the use of this paradigm in this environment can quantitatively improve the results of the project.

However, MDE in practical environment does not result too friendly for development teams. The concepts of models, metamodels, transformations, and among others, are not common notations for enterprise environment, and they seem too abstract and complex. However, we conclude that the use of UML profiles and UML-based tools offers an interface to deal with instances of metamodels quite suitable for analysts and designers and even for expert users.

Likewise, since transformations in QVT do not result easy to understand, our users do not work with it. They prefer an easier and more intuitive interface, as the NDT-Driver interface presented in Fig. 8.3, in order to benefit from its transformation power.

6 Future Work

NDT-Driver is a very powerful and useful tool to develop Web-oriented systems. However, we know that NDT-Driver is not functional enough. In this regard, we propose several improvements as future research.

Firstly, we suggest doing a research on how to improve test generation from functional requirements, particularly when functional requirements are described using activity diagrams. Currently, NDT-Driver generates all possible paths from the initial and final activities of the activity diagram of the functional requirement. As far as we know that the method is not suitable enough, we propose a research on how to enrich the transformation method with new techniques for the selection and reduction of redundant testing paths.

Secondly, another aspect that we must continue working on is considering other programming languages since today NDT-Driver is only available for Java.

Finally, we propose to investigate on how to incorporate heuristics in the generation of some models. For instance, in the generation of the navigation model of the analysis phase, NDT-Driver should identify aspects as that the graph is not connected. To achieve this, NDT-Driver could use the Warshall algorithm.

Acknowledgments This research has been supported by the project QSimTest (TIN2007-67843-C06_03), by the Tempros project (TIN2010-20057-C03-02), by the National Network Quality Assurance in Practise, and by the CaSA (TIN2010-12312-E) of the Ministry of Education and Science, Spain.

References

Ceri S, Fraternali P, Bongio A (2000) Web modelling language (WebML): a modelling language for designing web sites. Comput Netw 33(1–6):137–157

EA (Enterprise Architect) (2010). http://www.sparxsystems.com

Escalona MJ, Aragón G (2008) NDT: a model-driven approach for web requirements. IEEE Trans Softw Eng 34(3): 377–394

Escalona MJ, Gutierrez JJ, Villadiego D, León A, Torres AH (2007) Practical experience in web engineering. In: Magyar G et al (eds) Advances in information system development. New methods and practice for the networked society. Springer, New York

Fons J, Pelechano V, Albert M, Pastor O (2003) Development of web applications from web enhanced conceptual schemas. In: Song I-Y, Liddle SW, Ling TW, Scheuermann P (eds) Conceptual modeling - ER 2003. 22nd international conference on conceptual modeling, Chicago, IL, USA, 13-16 Oct 2003. Proceedings (LNCS 2813). Springer, Berlin/Heidelberg, pp 232–245

Koch N, Knapp A, Zhang G, Baumeister H (2008) UML-based web engineering. In: Rossi G, Pastor O, Schwabe D, Olsina L (eds) Web engineering: modelling and implementing web applications. Springer, London, pp 157–191

OMG (2008) Documents associated with meta object facility (MOF) 2.0 query/view/transformation. http://www.omg.org/spec/QVT/1.0/

Rossi G, Schwabe D (2008) Modelling and implementing web applications with OOHDM. Web engineering: modelling and implementing web applications. In: Rossi G, Pastor O, Schwabe D, Olsina L (eds) Web engineering: modelling and implementing web applications, Human-computer interaction series. Springer, London, pp 109–155

UML (2005) Unified modeling language: superstructure. Specification, OMG, 2005. http://www.omg.org/cgi-bin/doc?formal/05-07