Scientific Journal of Riga Technical University
Computer Science. Applied Computer Systems
2010
Volume 43

Testing and Traceability Aspects in the Context of the Model Driven Architecture (MDA)

Andrejs Grave, *Riga Technical University*

***Abstract* - With the growth of complexity of the software systems it becomes more complicated to ensure and evaluate quality of the software being built. This paper discusses quality of the software in the context of the Model Driven Architecture. Paper analyses factors that affect quality of the software in the software development projects that are developed using MDA. As one of the important factor that affects quality of the software, is traceability. This paper provides description of the traceability property and importance of it within development of the software. Within context of this paper traceability is considered as a property of a system description technique that allows changes in one of the system descriptions to be traced to the corresponding portions of the other descriptions. This paper is focused on such aspects of the software development as testing and traceability in the context of MDA. Paper contains in review of traceability, MDA and traceability within MDA. Also paper contains description of the method for formal definition of the problem domain – called Topological functioning modeling for model driven architecture (TFMfMDA). This paper introduces method of the application of the TFM as the traceability tool. TFM as the traceability tool can be used to analyze impact of the changes and select most important tests.**

Keywords: **model driven architecture, quality, testing, traceability**

I. Introduction

This work is based on the research on computation independent modeling (CIM) started in [1] – [4]. Above mentioned research introduce more formalism into problem domain modeling from a CIM viewpoint within OMG Model Driven Architecture (MDA) [8]. For this purpose formalism of a Topological Functioning Model (TFM) is used [1]. The main idea of the given work is to describe application of the topological functioning model in the context of the MDA from the traceability point of view. The TFM is a base of the suggested approach, i.e., Topological Functioning Modeling for Model Driven Architecture (TFMfMDA). TFMfMDA satisfies the main feature of MDA – Separation of Concerns. TFMfMDA provides functional requirements to be in compliance with functionality of the system under consideration at the very beginning of analysis. It makes it possible to use a formal TFM as a computation independent model without introducing complex mathematics.

Traceability is an important aspect of software development that is often required by various professional standards. Therefore this research analysis traceability within MDA, and possibility of establishing traceability links between computation independent model level and platform independent model level.

The goal of the research is to analyze applicability of TFM as traceability tool to ease software development and testing tasks and to identify fields which can be improved using TFM.

This paper describes theoretical method that could be used to apply TFM as traceability reference to solve such challenges as impact analysis, most important tests detection and test coverage monitoring.

This paper is organized as follows. Section II describes traceability in the software development – its definitions, motivation, difficulties. Sections III and IV discusses the MDA and the current state of the traceability within model driven development, and particularly in the context of the MDA. Section V describes principle of the topological functioning modeling and its application within MDA life cycle. Section VI discusses traceability between TFM and PIM level models. Conclusions section provide summary of the paper and establishes future works related with this research.

II. Traceability

The IEEE Standard Glossary of Software Engineering Terminology [5] defines traceability as follows:

"The degree to which a relationship can be established between two or more products of the development process, especially products having a predecessor-successor or master-subordinate relationship to one another; for example, the degree to which the requirements and design of a given software component match; (2) The degree to which each element in a software development product establishes its reason for existing; for example, the degree to which each element in a bubble chart references the requirement that it satisfies."

This definition is strongly influenced by the originators of traceability – the requirements management community. Gotel and Finkelstein [6] define requirements traceability as follows:

"… the ability to describe and follow the life of a requirement, in both a forward and backward direction; i.e., from its origins, through its development and specification, to its subsequent deployment and use, and through periods of ongoing refinement and iteration in any of these phases."

Researchers from IBM suggest a much broader definition of traceability. They regard traceability as any relationship that exists between artifacts involved in the software-engineering life cycle. Their definition includes, but is not limited to the following:

- Explicit links or mappings that are generated as a result of transformations, both forward (e.g., code generation) and backward (e.g., reverse engineering);

- Links that are computed based on existing information (e.g., code dependency analysis);
- Statistically inferred links, which are links that are computed based on history provided by change management systems on items that were changed together as a result of one change request [7].

The purpose of requirements traceability is:

- Demonstrate to the customer that the developed product conforms to the requested features, and that all changes arising throughout the development phases have been accounted for;
- Ensure that the test plan, test cases and test procedures in general are relevant to the requirements;
- Ensure that developers are not creating features that no one has requested.

From the above-mentioned points, it is clear to see that software development projects, must from inception, accommodate requirements traceability and retain the capacity to handle the frequently changing goals and needs of the user.

Motivation behind traceability contains following points:

- **To ensure completeness** – facilitate the identification of requirements which are not satisfied by the system by following traceability links;
- **To propagate the changes** – find out the elements impacted by changes at any time in the development process;
- **To facilitate mutual understanding** and **enable semantic interoperability** – as a lot of participants with a diverse background come in different development phases;
- **To manage information** produced during distributed collaborative system development.

Different stakeholders in the software development process have different traceability goals. The project manager perspective is that traceability supports demonstrating that each requirement has been satisfied and that each system component satisfies a requirement. From the perspective of requirements management, traceability facilitates linking requirements to their sources and rationales, capturing the information necessary to understand the evolution of requirements, and verifying that the requirements have been met. During design, traceability enables designers and maintainers to keep track of what happens when a change request is implemented before system is redesigned. With complete traceability, more accurate costs and schedules of changes can be determined, rather than depending on the programmer to know all the areas that will be affected by these changes.

Although the advantages are well documented, traceability practices are not widely used. The commonly stated reason is the high cost of manual creation and maintenance of traceability information. In addition, the lack of guidance as to what link information should be produced and the fact that those who use traceability are commonly not those producing it also diminishes the motivation of those who create and maintain traceability information.

III. Model Driven Architecture

At the conceptual level, MDA is a holistic approach to improving the entire software development life cycle – specification, architecture, design, development, deployment, maintenance, and integration – based on formal modeling.

More specifically, MDA is a framework of technical standards progressively being developed by the OMG – an open industry consortium supporting this approach – along with a set of usage guidelines for enabling the application of those standards with appropriate tools and processes.

According to the OMG, MDA exhibits the following characteristics: portability, cross-platform interoperability, platform independence, domain specificity, productivity [8].

Model-driven approaches to software and system development have been in use for some time. More recently, the focus of attention has been on how to increase automation of model-driven technologies based on the OMG's MDA approach. MDA techniques enable organizations to construct custom automations for model-to-model and model-to-code transformations. Using these transformations, technical experts can share their expertise across a large development team. In particular, MDA offers a number of advantages over other approaches:

- Productivity of development is increased by helping to insulate a majority of developers from technical details that they do not need to consider to carry out their task of designing enterprise solutions to meet a business need. More time is spent focusing on their task at hand: implementing the business capabilities of the system;
- MDA improves quality by encouraging reuse of known patterns of behavior, building on existing architectural designs, and leveraging expertise more effectively. The use of automation also promotes consistency and improves the quality of the system design and implementation, particularly as the solution evolves through maintenance and upgrade;
- MDA's repeatable set of practices is well-suited to today's iterative development requirements, because MDA enhances predictability of the development process and the delivered solution itself. The use of automation speeds up development, particularly when many of the tasks of the developer are repetitive and cumbersome.

The essential characteristics of MDA, then, are:

- Separation between, and reusability of, domain and platform expertise;
- Reuse and leveraging of existing IT technologies;
- Quick adaptability of domain and technology changes;
- Generation of working, high-quality applications and integrations.

The model-driven approach focuses on models as the primary artifacts. MDD recognizes the necessity of having several kinds of models to represent the system as it progresses from early requirements through final implementation. These models may represent different aspects of the system (e.g., structural or behavioral), or they may represent the system at varying levels of abstraction (e.g., an analysis model or a design model). In this section, we discuss

the role of traceability in maintaining consistency among models and in model transformations.

MDA, the Object Management Group approach to MDD, is based on modeling and automated mapping of models to implementations through model transformation. Two standardization efforts are underway to realize model transformations: the MetaObject Facility (MOF) 2.0 Query/Views/Transformations (QVT) [9] and the MOF Model-to-Text Transformation Language [10]. Transformations may be unidirectional or bidirectional. Updating unidirectional transformations and synchronizing bidirectional transformations require the means to identify the existing target model element related to a given source model element. Thus, such transformations need to create and maintain mappings between source and target models.

Technologies to perform model transformations range from conventional programming languages to specific transformation languages [11]. Czarnecki and Heslen [12] propose a taxonomy for the classification of model transformation approaches. The generation of traceability links is one of the dimensions of this classification. According to Czarnecki and Heslen, transformation approaches divide into those that provide dedicated support for traceability and those that depend on the user to encode the generation of traceability, using the same mechanisms as those for generating any other kinds of model elements. In the case of automated support, the approach may still provide some control over how many traceability links are created (in order to limit the amount of traceability data). Finally, there is the choice of the location at which the links are stored, for example, in the source and target, or separately.

IV. Traceability Within MDA

Model driven development and as its case MDA, places new demands on such traceability tools, which must now be capable of dealing with different types of models (for example business, design, architecture, deployment) and their attributes [7].

In addition, with evolving software development and communication ways, tools must support collaborative development properties, such as e-mails, chat discussions, etc.

Inconsistencies among models representing different viewpoints of a system, or among specifications at differing levels of abstraction, can arise during or between phases of software development, raising the compelling issue of how to manage consistency among different models and between models and code [13]. Grundy et al. [14] provide an excellent review of various tools and different approaches to inconsistency management. According to their view, some inconsistencies may be automatically corrected; however, many inconsistencies cannot, or should not, be automatically corrected. Hence, mechanisms are required to inform developers of inconsistencies and facilitate the monitoring and resolution of inconsistencies.

Desfray believes that traceability is an essential part of the solution [15]. Several recent papers have proposed to use traceability as the basis for detecting and informing related stakeholders about inconsistencies [16] – [18]. As it is usually impossible to keep a software system consistent at all times, tools need to have different policies for consistency enforcement. Future work in this area is needed to define a traceability model that supports detecting, representing, storing, and propagating a wide range of inconsistencies, and tying these with appropriate inconsistency management policies.

Jouault [19] distinguishes between internal traceability, which is maintained by the transformation engine during the transformation to assist with the transformation algorithm, and external traceability, which is a persistent mapping kept beyond the transformation execution. Many approaches allow the serialization of internal traceability information. The main drawback is that the format and granularity in which such traceability is stored are predefined and may not be compatible with the representations used by other tools that make use of traceability information.

Although much progress has been done in the area of model transformations, the integration with traceability is still weak. Transformation solutions tend to define their own internal traceability and do not integrate well with existing traceability solutions that provide analysis and query capabilities on the traceability information.

There are following problems in requirements traceability (these are true for both – model driven development methods and for non-model driven or traditional methods):

- Need for guidelines on what types of information must be captured and used in what contexts;
- The interpretation of the meanings of linkages between system components is left to the user;
- Need for abstraction mechanisms to allow variation of granularity (aggregation) and sophistication (generalization) in traceability tasks;
- Need for inference services supporting the semantic of the different traceability link types;
- Need for mechanisms that allow project managers and developers to define and enact a model driven trace process accompanying the development process.

V. TFMfMDA

This section discusses the proposed TFMfMDA approach [1], [2], [20]; the main steps discussed further in the paper are illustrated by bold lines in Fig. 1. The TFMfMDA uses some capabilities of the universal category logic and is based on the formalism of the Topological Functioning Model.

The TFM is an expressive and powerful instrument for a clear presentation and formal analysis of functionality of a system and the environment the system works within.

The methodology of topological modeling of system functioning (TFM) was developed at Riga Technical University. First time its theoretic foundations were represented by Janis Osis in [1]. Its applications in different areas are being developed today as well.

The topological functioning modeling is a formal approach, which is based on assumption that a complex system can be described in abstract terms as a topological space (X, Θ),

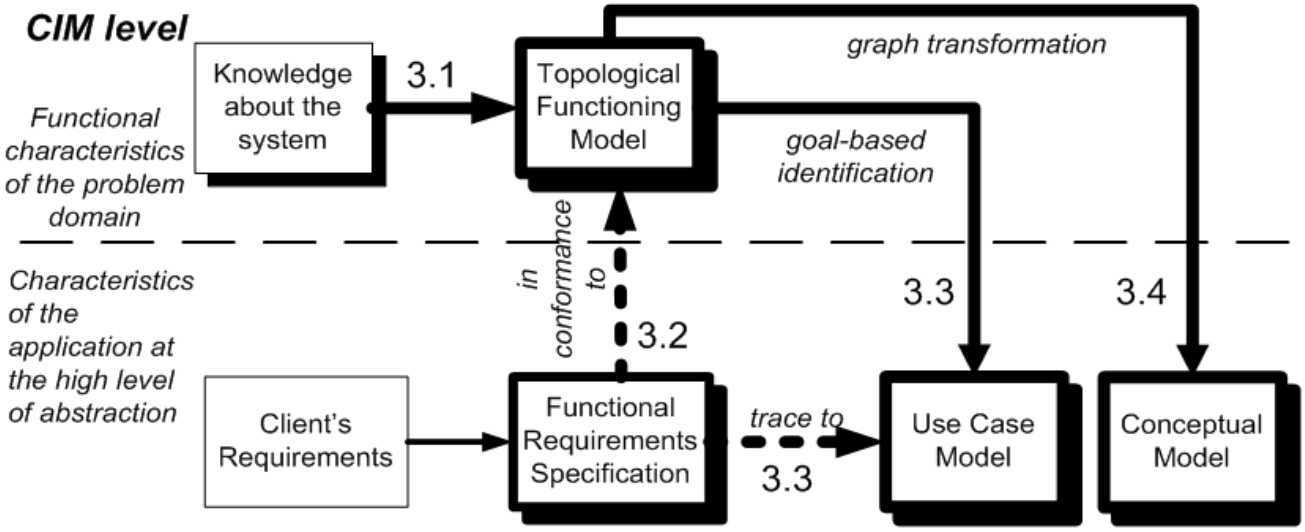

Fig. 1. Creation of the CIM using TFMfMDA

where X is a finite set of properties or functional features, and Θ is a topology in the form of a directed graph.

Functional feature represents:

- An action of the object;
- A result of the action (optional);
- An object that get the result of the action or an object that is used in this action; it could be a role, a time period or a moment, catalogues etc.;
- A precondition or an atomic business rule (optional);
- A system responsible for its execution (optional).

Each functional feature could be represented in the following form:

```
<action>-ing the <result> [to, into, in, by, of,
from] a(n) <object>
```

Example of the functional feature: "*Receiving a book from the reader*."

Functional feature is a characteristic of the system (in its general sense) that is designed for achieving some system's goal [1], [2].

J. Osis in [1] suggested the formal method for construction of a topological model of functioning of both technical and business systems from its informal verbal description. It is as follows:

1. Define a set X of physical or business functional characteristics of the considered system and other systems interacting with the system itself;
2. Introduce in X a topology in the form of a digraph G(X, V) indicating the cause-and-effect relations existing among elements of X (where V is a set of directed lines connecting elements of X);
3. Topological functioning model separation from a constructed topological space.

Necessary condition for the construction of such topological space G(X, V) is a meaningful and exhaustive verbal, graphical, mathematical description of the system. The adequacy of the model to describe the functioning of a concrete system can be then achieved by analyzing the mathematical properties of such an abstract object.

There are two kinds of information at the beginning of the problem analysis (the dashed line in Fig. 1 shows **Separation of Concerns**): The first one is at the (business or enterprise) system level, and the second one is at the application level. The main idea is that functionality determines the structure of the planned system (Fig. 1). Having knowledge about a complex system that operates in the real world, a topological functioning model of this system can be composed (Fig. 1, link 3.1). This means that a TFM of the system is tested and can be partially changed by functional requirements (Fig. 1, link 3.2). Then TFM functional features are associated to business goals of the system; this provides business use case as well as system use case identification according to the problem domain realities. Moreover, after those activities functional requirements are not only in conformance with the business system functionality but can also be traced back to the system use case model (Fig. 1, link 3.3). Problem domain concepts are selected and described in an UML Class Diagram (Fig. 1, link 3.4).

A TFM has topological (connectedness, closure, neighborhood, and continuous mapping) and functional (cause-effect relations, cycle structure, and inputs and outputs) characteristics. It is acknowledged that every business and technical system is a subsystem of the environment. Besides that a common thing for all system (technical, business, or biological) functioning should be the main feedback, visualization of which is an oriented cycle. Therefore, it is stated that at least one directed closed loop must be presented in every topological model of system functioning. It shows the "main" functionality that has a vital importance in the system's life. Usually it is even an expanded hierarchy of cycles. Therefore, a proper cycle analysis is necessary in construction of the TFM, because it enables careful analysis of

system's operation and communication with the environment [3].

VI. TFM AS TRACEABILITY AND TESTING TOOL

This section discusses traceability aspect of the TFMfMDA approach that was described in the previous section. TFMfMDA prescribes that TFM in the context of MDA should be used as a CIM. In the PIM level most widely are used UML diagrams (use case diagram, class diagram, activity diagram etc.), thus to establish traceability between CIM and PIM, traceability between TFM and UML diagrams need to be established (Fig. 2). Such traceability link will facilitate consistency between system design (PIM level UML diagrams) and business system functioning as it is (CIM level TFM).

The main idea is to connect functional features of the system TFM and UML artifacts in the PIM, in such way that every functional feature in TFM have corresponding ("is implemented by") element in PIM level (class, action, use case etc.). As it is shown in the Fig. 2, from "Business process" and "Environment" of the system is created TFM, which describes functioning of the system from computation independent manner. After TFM is created traditional MDA process could follow, but while working on the PIM level artifacts system designers should record traceability information – which particular functional feature of the TFM is described in the PIM. After that usual MDA process proceeds.

If such traceability links are collected and managed, this information facilitates following aspects of the software development:

- Consistency check between information system design and system functioning can be preformed formally;
- As TFM is a cause-effect model it can be easily used to identify parts affected by the change;
- Identification of most important tests (analyzing cycles, functional features influenced by the external systems);

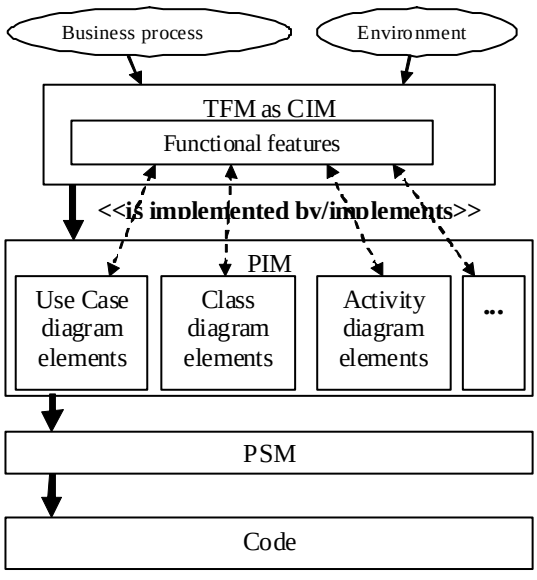

Fig. 2. Application of the TFM within MDA

- As the TFM contains not only business system itself, but also environment that has impact on the system, thus IS development can react on environmental changes in time;
- As TFM shows *ALL* system context we can use it as the tool for testing coverage analysis.

On this stage of the research only one type of traceability link is used between functional feature of the TFM and UML element – "implemented by", which binds together functional feature and elements that implement it.

By the comparatively simple form TFM ensures common understanding of the whole system and environment that has impact on the system. But common understanding of system functioning is essential to successful development of the information system. By its simple form TFM can be used as the communication tool within project context, it can be used to discuss changes with domain experts, business analysts, software developers and testers, thus ensuring that all project participants have common understanding of the system functioning. In addition topological functioning model while created can serve as the knowledge gathering tool. While creating TFM, modeler must gather and combine information from various sources – system descriptions, documents, domain experts, observations etc. By the cause and effect links, TFM development process drives creator to complete full picture of the system functioning.

First way in which described traceability link would help is change impact analysis. In the software development different kinds of changes can occur: business changes (change request), refinement, bugs, new functionality, and adaptation to some environmental. Software development process must support changes, not disturb these. MDA is one of those software development paradigms, which allows changes, and allows fast implementations of these changes.

But with every change developers must consider if they have implemented change correctly, and that they didn't broke other parts of the system. This question is addressed to change impact analysis. The goal of impact analysis is often to determine what would be affected by a change to a particular artifact. This would involve the following logical steps:

- Identify the artifact that is to be modified (or represents the service to be modified);
- Trace back through any relationships that indicate a dependency on that artifact. These relationships will have the selected artifact as a target of the relationships. This impact analysis thus results in a list of "dependent artifacts" that depend directly on the selected artifact;
- If a change affects these "dependent artifacts", then other artifacts that depend on these "dependent artifacts" will also be affected. The impact analysis must therefore act recursively looking for relationships that have any of the "dependent artifacts" as targets;
- This process continues until a complete list is obtained starting at the selected artifact and finishing with artifacts on which nothing else depends. Each artifact in the list could therefore be affected by the change to the selected artifact.

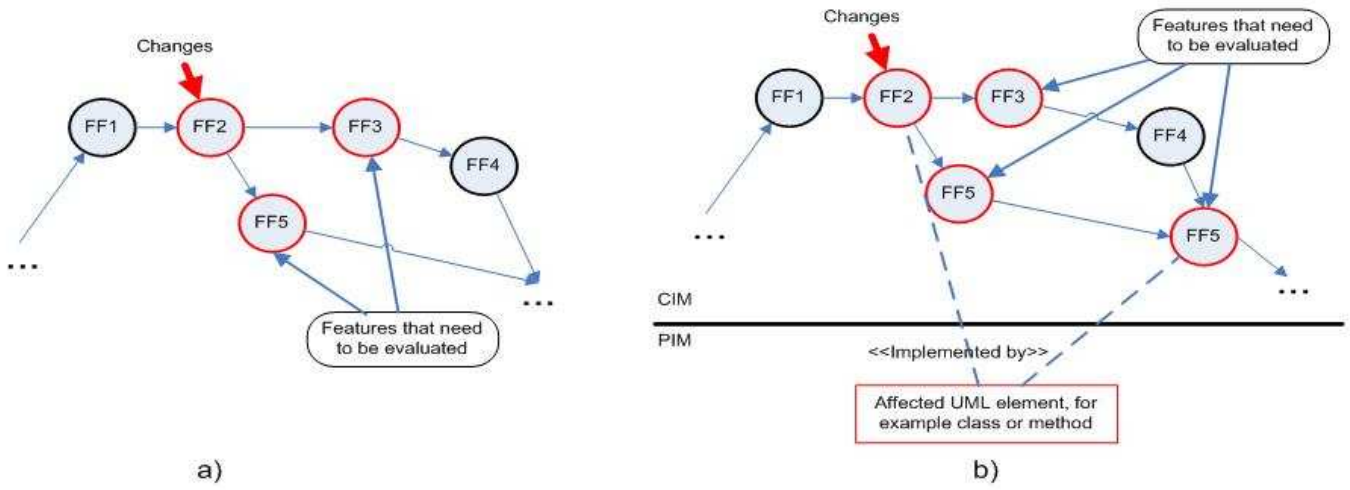

Fig. 3. Examples of impact analysis

By the means of cause and effect nature of the TFM, it provides effective way to find impacted parts of the system in the case of changes. Effective in this case means, that cause and effect relations between functional features are already present in the TFM, thus there is no need to gather this information separately.

TFM provides information of the system usage (how software system is functioning, and that provides extra information when evaluating change impact). Modern IDE's are able to provide developer with tracing information such as were some method is called, or where particular class is used, but these IDE's cannot automatically provide developer with information in which order functions can be called by a user of the system. But such kind of information is quite useful in the development (change impact analysis, debugging, etc.) process.

Some examples of the impact analysis can be seen on Fig. 3:

a) Assume functional feature "FF2" need to be changed, basis on this we need to evaluate functional features "FF3" and "FF5" if these features need to be aligned accordingly to "FF2" change;

b) This situation is very similar to situation a), but in this situation we have available tracing link of the functional feature "FF2", and we see that we need to evaluate functional feature "FF5" also, because it depends on the same implementation elements as functional feature "FF2".

Besides relating functional features with corresponding design elements, we can link functional features with testing information, such as: test cases, testing execution results and bug reports (see Fig. 4). Such relations would provide following advantages:

- It would be easier to analyze coverage of the test cases;
- Identification of the test case priority and order of the test case execution;
- Evaluation of the testing results in easier way (visual presentation of the testing status).

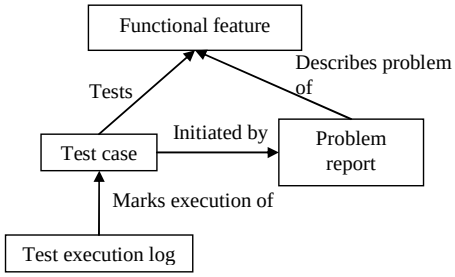

Fig. 4. Functional feature relation to the testing artifacts

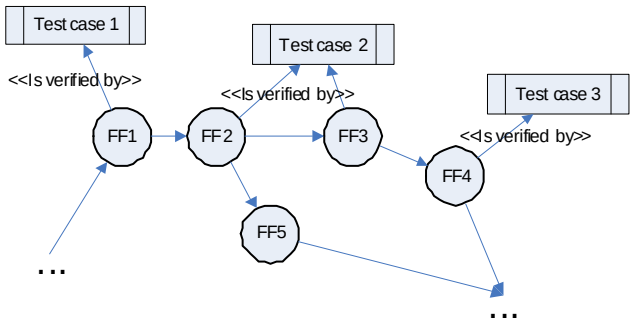

Fig. 5. Topological functioning model and corresponding test cases

To ensure complete testing of the system, all the test cases that are produced should be linked with corresponding functional features, and functional feature chains, that they validate. In such way effective test management information can be gathered. In such way tester can easily identify functional features that are not validated by any tests, e.g. functional features that are not tested. For example (Fig. 5), as we can see from the picture functional features "FF5" doesn't have any related test case, which means that this functional feature is not tested.

As it is possible to identify functional features that need to be changed or evaluated, it is also possible to identify test cases that need to be executed when necessary changes are implemented.

Other way that TFM helps testing is identification of most important tests. Most important tests must test functional features that: 1) are in the main cycle of the TFM of the system; 2) that communicate external environment or external systems; 3) are most used in the TFM (that have most incoming relations).

TFM can also provide source information to discover an order in which tests should be prepared and order in which system functional features should be tested.

Testing status can be measured and evaluated within context of the functional feature – how many tests are related to functional feature and how many of these tests are successfully completed. Such information can be collected and maintained for every functional feature. In such way it is possible can clearly understand which parts of system are tested and which are not, which parts of system work well and which don't. Information regarding testing results can be presented in the visual manner, for example functional feature can colored red, green or yellow depending on the state of testing result (related bug, successful/unsuccessful test cases etc.). Such way of presenting testing data allows easily understanding, evaluating and communicating testing information and status.

VII. Conclusions and Future Work

This paper describes the way of using topological functioning model within MDA life cycle to improve traceability and testability of the system under development. The utilization of the TFM approach can facilitate traceability and provide an effective basis for testing within MDA. First sections introduce readers with research context – MDA, traceability, TFM. After that method of the application of the topological functioning model as the traceability tool is described. Topological functioning model provide a unifying and often simplifying concept for a system and can drive designing, development, testing and documentation effort. Main areas, where topological functioning model, can give improvement are:

- Change impact analysis;
- Evaluation and planning of the changes;
- Selection of the most important tests;
- Identification of the current state of the testing.

Future research will contain following topics:

- Identification of the most valuable tracing links between TFM and PIM levels;
- Identification of the possible technological solutions for the proposed method;
- Application of the proposed method to solve real world problem (case study);
- Examine possibility of application of the proposed approach on existing systems where TFM is not used from the beginning.

References

[1] J. Osis, "Formal computation independent model within the MDA life cycle," *International Transactions on Systems Science and Applications*, vol. 1, no. 2. Glasgow, UK: Xiaglow Institute Ltd, 2006, pp. 159-166.

[2] E. Asnina, "Formalization aspects of problem domain modeling within model driven architecture," in *Databases and Information Systems, 7th International Baltic Conference on Databases and Information Systems, Communications*, Vilnius, Lithuania, Materials of Doctoral Consortium. Vilnius: Technika, 2006, pp. 93-104.

[3] J. Osis, "Software development with topological model in the framework of MDA," in *Proceedings of the 9th CaiSE/IFIP8.1/EUNO International Workshop on Evaluation of Modeling Methods in Systems Analysis and Design (EMMSAD'2004) in connection with the CaiSE'2004*, Riga, Latvia, Vol. 1. Riga: RTU, 2004, pp. 211-220.

[4] J. Osis, E. Asnina, and A. Grave, "Formal computation independent model of the problem domain within the MDA," in *Proceedings of the 10th International Conference on Information System Implementation and Modeling*, Hradec nad Moravicí, Czech Republic, April 23-25, 2007, pp. 47-54.

[5] *IEEE Std 610.12-1990. IEEE Standard Glossary of Software Engineering Terminology*. New York: IEEE, September 1990.

[6] O. C. Z. Gotel and A. C. W. Finkelstein, "An analysis of the requirements traceability problem," in *Proceedings of the First International Conference on Requirements Engineering*, Utrecht, The Netherlands, 1994, pp. 94-101.

[7] N. N. Aizenbud-Reshef, B. T. Nolan, J. Rubin, and Y. Shaham-Gafni, "Model traceability," *IBM Systems Journal*, vol. 45, no. 3, 2006.

[8] "MDA guide version 1.0.1", Object Management Group. [Online]. Available: http://www.omg.org/cgi-bin/doc?omg/03-06-01.pdf. [Accessed: 1.07.2010.].

[9] "Revised submission for MOF 2.0 Query/View/Transformations RFP (ad/2002-04-10)," QVT-Merge Group, Version 2.1, OMG Document ad/05-07-01, Object Management Group, Inc., August 2005.

[10] "MOF model to text transformation language RFP," OMG Document ad/2004-04-07, Object Management Group, Inc., revised on May 27, 2004.

[11] J. Bézivin, B. Rumpe, A. Schürr, and L. Tratt, "Model transformations in practice," in *ACM/IEEE 8th International Conference on Model Driven Engineering Languages and Systems*, Montego Bay, Jamaica, October 2005. [Online]. Available: http://sosym.dcs.kcl.ac.uk/events/mtip05/long_cfp.pdf.

[12] K. Czarnecki and S. Helsen, "Feature-based survey of model transformation approaches," *IBM Systems Journal*, vol. 45, no. 3, 2006, pp. 621-646

[13] R. Paige and Y. Shaham-Gafni, "Model composition: development of consistency rules," Modelware Report D1.5, September 2005. [Online]. Available: http://www.modelware-ist.org.

[14] J. Grundy, J. Hosking, and W. B. Mugridge, "Inconsistency management for multiple-view software development environments," *IEEE Transactions on Software Engineering*, vol. 24, no. 11, pp. 960-981, November 1998.

[15] P. Desfray, "MDA – when a major software industry trend meets our toolset, implemented since 1994," White paper, Softeam, 2001. [Online]. Available: http://www.omg.org/mda/mda_files/MDA-Softeam-WhitePaper.pdf.

[16] J. Huang-Cleland, C. K. Chang, and M. Christensen, "Event-based traceability for managing evolutionary change," *IEEE Transactions in Software Engineering*, vol. 29, no. 9, September 2003, pp. 796-810.

[17] T. Olsson and J. Grundy, "Supporting traceability and inconsistency management between software artifacts," in *Proceedings of the IASTED International Conference on Software Engineering and Applications*, Boston, MA, November 2002.

[18] N. Aizenbud-Reshef, R. F. Paige, J. Rubin, Y. Shaham-Gafni, and D. S. Kolovos, "Operational semantics for traceability," in *ECMDA Traceability Workshop*, Nuremberg, Germany, November 2005, pp. 7-14.

[19] F. Jouault, "Loosely coupled traceability for ATL," in *Proceedings of the European Conference on Model Driven Architecture Workshop on Traceability*, Nuremberg, Germany, November 2005. [Online].

Available: http://www.sciences.univ-nantes.fr/lina/atl/www/papers/ECMDATraceability05.pdf.

[20] E. Asnina, "Formalization of problem domain modeling within model driven architecture", PhD thesis. Riga, Latvia: RTU Publishing House, 2006.

Andrejs Grave was born and grew up in Daugavpils, Latvia. There Andrejs finished Daugavpils Experimental School and after that relocated to Riga to study in the Riga Technical University. Andrejs Grave has a B.Sc. in Computer Sciences (Riga Technical University, Latvia, 2003), and Mg.Sc.ing. in Computer Sciences (Riga Technical University, Latvia, 2005). Since 2006 he is PhD (Riga Technical University, Computer science faculty) student with research interests in the model-driven software development and testing.

Andrejs started his professional career started in 2002 as PL/SQL Trainee Developer in Social State Insurance Agency. During work career in Social State Insurance Agency He has worked in following positions (in chronological order): PL/SQL Developer, Oracle Reports Developer, Lead PL/SQL Developer, Lead Tester, System Analyst. Since May of 2009 Andrejs is working as Java Developer in C.T.Co (15/25 Jurkalnes st., Riga, Latvia). Since January of 2007 He also works as Researcher in the Riga Technical University, Faculty of Computer Sciences (1/4 Meza st., Riga, Latvia).

Andrejs Grāve. Testēšanas un trasējamības aspekti modeļu vadāmās arhitektūras kontekstā

Pieaugot informācijas sistēmu sarežģītībai, arvien sarežģītāk kļūst novērtēt un nodrošināt programmatūras kvalitāti. Dotais pētījums apskata programmatūras kvalitātes aspektu Modeļvadāmās arhitektūras kontekstā. Rakstā tiek analizēti faktori, kas ietekmē programmatūras kvalitāti, kas tiek izstrādāta vadoties pēc Modeļvadāmās arhitektūras principiem. Kā viens no svarīgiem faktoriem, kas ietekmē programmatūras izstrādes kvalitāte tiek apskatīta trasējamība. Dotais raksts apraksta trasējamības īpašību un tās svarīgumu programmatūras izstrādes procesā. Dotā raksta kontekstā trasējamība tiek aplūkota kā sistēmas apraksta tehnikas īpašība, kas ļauj vienā sistēmas aprakstā veiktās izmaiņas ,attiecināt uz atbilstošām daļām citos sistēmas aprakstos. Dotajā rakstā uzmanība tiek vērsta uz tādiem programmatūras izstrādes aspektiem, kā trasējamība un testēšana. Dotajā rakstā piedāvāts lietot topoloģisko funkcionēšanas modeli trasējamības kvalitātes veicināšanai. Sasaistot topoloģisko funkcionēšanas modeli (TFM) ar no platformas neatkarīgo modeli (PIM), tiek atvieglota un uzlabota izmaiņu ietekmes novērtēšana un svarīgāko testu noteikšana.

Андрей Граве. Тестирование и трассируемость в контексте МДА

Информационные системы становятся все сложнее, и вместе с тем все сложнее оценить и обеспечить качество программного обеспечения. Данное исследование рассматривает качество разрабатываемого программного обеспечения в контексте МДА. В статье анализируются факторы, которые влияют на качество разрабатываемых информационных систем. Как один из важнейших факторов рассматривается свойство трассируемости. В контексте данного исследования трассируемость рассматривается, как свойство техники описания системы, которое позволяет связать изменения сделанные в одном описании системы, с соответствующими частями в других описаниях системы. В данной статье внимание уделяется трассируемости и тестированию информационных систем разработанных используя МДА. В данной статье предлагается использовать топологическую модель функционирования системы, как инструмент для трассирования. Использование топологической модели функционирования системы для трассировки позволяет лучше решить такие задачи как анализ воздействия изменений и отбор самых важных тестов.