Abstract. Computer systems and their specifications often require modifications during software development and maintenance processes. The usual thing is when changes in software are not reflected in the corresponding specifications. This makes change verification difficult, because of unpredictable side effects after implementation. Model Driven Architecture and automated transformations should make it easier to verify changes and to predict side effects by using trace links among development artifacts. However, the trace links among software functional units and functional units of the system where this software will work as well as impact of software changes to this environment are implicit and intuitive. Formalization of specifications of the environment and software functionality as well as their analysis by means of Topological Functioning Model enables visualization of the trace links. This paper demonstrates the establishment of formal trace links to business functional units and entities from user requirements and analysis artifacts, which show element interdependence explicitly.

Keywords: requirements traceability, software changes, topological functioning modeling, Model Driven Architecture

1 Introduction

As one of Lehman’s five laws of the evolution of a software system states, the software system is a subject to non-stop changes, otherwise it becomes less and less useful for its initial purpose (as cited in Yang & Ward, 2002). In spite of the variety of development means, software systems require frequent modifications even in our days. The common issue for software systems is their long-lasting inconformity to the external environment, where the system will work, and user requirements. Frequent change implementations lead to breaking structure of the software system and appearance of unpredictable side effects. Unpredictability of side effects is a result of the traditional development process, where the up-to-date documentation is mainly code and logical links, trace links, among functional units are not evident at once.
Therefore, establishment and management of the trace links among functional as well as logical units in software and specifications is very important.

This paper discusses application of Topological Functioning Model (TFM) for functional requirements traceability needs. We demonstrate how formalization of a computation independent model (CIM) used within Model Driven Architecture (MDA) by means of functional and topological properties of the TFM helps in establishing the trace links among artifact units, predicting side effects and decreasing change implementation issues when functional requirements are modified. This research continues research presented in (Osis, 2006; Osis & Slihte, 2010; Osis & Donins, 2010; Asnina & Osis, 2010; Asnina & Osis, 2011).

The paper is organized as follows. Section 2 briefly introduces categories of changes and software maintenance, discusses an effect of changes on MDA models in dependence of their abstraction level. The related work also is discussed here. Section 3 describes and illustrates how the suggested formalization of the CIM can be applied in verification of changes. Section 4 demonstrates the proposed idea by the example of a library system. Conclusion discusses results and future research directions.

2 Model Driven Architecture, Changes and Traceability

Software evolution is a permanent process. According to categories of changes, software modification activities can be divided into the following corresponding categories (Yang & Ward, 2002):

- Corrective maintenance. This case considers removing a possible fault in the software, which indicates on software system nonconformity to its specification, or on incomplete or inconsistent specification.
- Adaptive maintenance. This case considers system modification as a result of changes to the external environment in which the system operates (e.g., hardware).
- Perfective maintenance. This case considers system modification as a result of changes in user requirements (e.g., caused by changes in business operation).
- Preventive maintenance. This case includes system modification in order to foresee future problems and make subsequent maintenance easier.

The cause of side effects in coding, data and documentation still are complexity of large software systems, a lack of conformity to complex human institutions and systems, implementation of changes not presented in up-to-date documentation, and invisibility of links among software system parts in complete and clear graphical representation. Requirements traceability is a means for analysis and graphical representation of logical links as well as verification of changes.

As Gotel and Finkelstein (1994) wrote “requirements traceability is an ability to follow the life of a requirement”. They distinguished two directions of requirements traceability – pre-RS (backward) and post-RS (forward). Pre-RS traceability refers to requirement’s life prior to its inclusion in the requirements specification. Post-RS traceability refers to requirements life from the moment of its inclusion in the requirements specification. However, one of the main issues in traceability practice is
the high cost of manual creation and maintenance of traceability information (Aizenbud-Reshef, Nolan, Rubin, & Shaham-Gafni, 2006).

Theoretically, in case of correct and entire implementation, the idea and principles of MDA proposed by the Object Management Group (OMG) are able to solve some of these issues, e.g., links among software parts and side effects of changes theoretically could be automatically managed and visible in always up-to-date documentation – software system models; moreover, these links would describe the state of the software system completely and clearly.

There are three main models in MDA: a computation independent model (CIM), a platform-independent model (PIM) and a platform-specific model (PSM). They model system at different levels of abstraction. Besides these main models, additional models may also be used – they are a transformation model, a traceability model, a platform model, a database model, and so on. In case of automated transformations, the traceability model can be created automatically because trace links among artifact units are explicit. In this paper, we limit ourselves with the three main models of MDA as well as code and transformations.

Considering the categories of changes (corrective, adaptive, perfective and preventive), we can define that changes may occur in the external environment and knowledge about the problem domain, user requirements, software architecture, platform-specific details – software technology or hardware used, and code itself.

![Diagram showing MDA models and categories of changes](image)

**Fig. 1.** Categories of changes and their impact on MDA models

Within MDA (Fig. 1), adaptive and perfective changes should be first verified and implemented in the CIM, and only then propagated to PIM to PSM to code; preventive changes affect software design already specified in the PIM, therefore they must be verified on compliance with the CIM and propagated from [CIM to] PIM to PSM to code; in turn, corrective changes can affect every MDA model.

Certainly, the most significant case is when incompilance is located in the requirements specification. In this case, modifications start from the CIM and are
Erika Asnina, Bernards Gulbis, Janis Osis, Gundars Alksnis, Uldis Donins, Armands Slihte

propagated to code through PIMs and PSMs. Post-RS traceability seems is able to help in change requests verification and change cost estimation. The positive fact is that post-RS traceability information is kept together with transformations (that are to be automated starting from the PIM level). However, post-RS traceability is not able to help in identifying the origin (and production) of requirements.

The current literature contains ample publications describing support of post-RS traceability, rather manually defined than defined by using heuristic traceability rules in the stage from requirements (CIM) to analysis models (PIM). However, most approaches lack support of pre-RS traceability (within CIM). Problems confronting pre-RS traceability are enumerated in detail in (Gotel & Finkelstein, 1994).

Pre-RS traceability not only includes a documented history of eliciting the final requirements from a pre-requirements set (Grammel & Kastenholz, 2010), but also a stakeholder aspect, which is oriented on dependencies between high-level stakeholders (as well as organizations, system missions, standards) as a starting point for driving and documenting requirements process (Sahraoui, 2005). However, MDA either considers the CIM only as informal (textual) requirements model without its relating to business and knowledge models, thus the high-level stakeholder aspect is often skipped, or uses traditional requirements engineering models/languages.

A very ambient overview on recent publications about requirements generation from software engineering models (and thus possibility to trace their origin) is performed by Nicolás and Toval (2009). Software models used are KAOS, i* and temporal logic goal-oriented models, RAD business models, eEPC business models as well as use cases and scenarios, UML models and user interface. Models are ordered by the decreasing number of applications. As the authors concluded, the better way is to use natural and formal languages together. High-level business process models in BPMN notations and their transformations also can be used (De Castro, Marcos, & Vara, 2011).

Other way to improve traceability of knowledge back to its initial acquisition is the tagging concept introduced by Richter, Abowd, Miller and Funk.

Summarizing, such issues as a lack of conformity to complex human institutions and systems and misunderstanding of the system’s purpose between developers and users of the software system could not be solved only by strong conformity between system requirements, PIMs, PSMs and code. This case requires the pre-RS traceability in a broad sense. There must be up-to-date formal models that contain knowledge about the domain (about those human institutions and systems, where the software systems work or will work) at the computation independent level. Moreover, system requirements must be traceable, i.e., in strong conformity with these formal models. Requests for changes which affect the computation independent level, first, must be traced backward and verified on the conformity to domain knowledge and correspondence to system’s purposes/missions/standards/functionality, and only then traced forward and implemented in models at the lower levels of abstraction and code. This should allow at least predicting, and at most avoiding, side effects of change implementations. We discuss application of our proposed formal model in the next sections.
3 Changes and Requirements Traceability within Topology-Based MDA Lifecycle

The Topological Functioning Model, TFM, is a formal mathematical specification of domain functioning. Its mathematical foundations, topological and functioning properties, are described in detail in (Osis & Asnina, 2011). The TFM introduces modification in the beginning of MDA development life-cycle (Osis, 2006) – in the CIM. It captures business knowledge about the problem domain, i.e. business functional characteristics and cause-and-effect relations among them, organizational units and roles and their responsibility for providing and using those functional characteristics, and domain objects and their participation in business functioning (Asnina & Osis, 2011). Moreover, the TFM is a ground for checking compliance of user requirements and software functional requirements to the problem domain (Osis & Asnina, 2008).

Fig. 2 illustrates the formalized part of MDA software development lifecycle, the CIM, by means of the TFMs of the business system and information system.

Fig. 2. Changes and requirements traceability at the beginning of topology-based MDA software development lifecycle

The TFM of the business system (BS) formally specifies functionality of the corresponding human institution or system. In turn, the TFM of the information system (IS) formally specifies functionality of the computer system that will be or is already used in that human institution or system. These two models are continuously mapped, i.e., they are formally interrelated and all changes in one model must be specified in the other one. Transformation and traceability implementation between models is formal and based on mathematics – continuous mapping of graphs. Properties of the continuous mapping between TFMs are explained in detail in (Asnina & Osis, 2010).

The construction of the TFM of BS is described at large in (Osis & Asnina, 2011). Fig. 2 points to the fact that construction of the TFM of the system is manual activity that transforms informal verbal descriptions to the formal mathematical specification. However, first positive results of the on-going research on automation of construction
Erika Asnina, Bernards Gulbis, Janis Osis, Gundars Alksnis, Uldis Donins, Armands Slihte

of the TFM of BS by using a natural language processing system are demonstrated in (Osis & Slihte, 2010).

Initially, models of BS and IS are equal. Verification of user requirements (desires) based on formal mappings from them to this initial TFM of IS results in checking and correcting requirements as well as refining and modifying the TFM of IS, while keeping its formal consistency with the TFM of BS. Then formal derivation of software functional requirements as well as entities (classes) in domain vocabulary becomes possible.

Another manual activity demonstrated in Fig. 2 is transformation from CIM to PIM, i.e., from domain vocabulary and system requirements to analysis models of system structure and behavior. The initial results of the ongoing research on formalization of this informal activity by means of the TFM are illustrated in (Osis & Donins, 2010). However, even these first results show that formal tracing from the TFM of IS, domain vocabulary and software functional requirements to the analysis model is possible.

A general Pre-RS traceability framework is illustrated in Fig. 3. Classes at the conceptual level, actors and use cases (as software functional requirements), functional requirements and system goals set by users (users’ desires) are traced to TFM functional features, traceability between which at different levels of abstraction is also provided. Hence, it is possible to determine why a class or requirement is included in the software solution, and how it is linked with other structural or behavioral elements by using the formal ground – formally related topological models of business and information systems.

4 Pre-RS Traceability in case of the Perfective Change Request

For illustration of advantages of the introduced formalism, let’s take a little bit simplified example that describes an information system of the library “Library IS”.

![Fig. 3. Pre-RS traceability framework in general](image-url)
4.1 The CIM – Business Model


User requirements (functional) are dedicated to receiving an IS that supports servicing readers, i.e. the following user requirements (desires) for the IS are set - FR1: The system shall register a new reader; FR2: The system shall check out a book; FR3: The system shall handle return of a book copy; FR4: The system shall account reader’s fines.

Tracing: The requirements map onto functional features of the TFM of BS as follows (Fig. 4b): FR1 to {2, 3, 4}, FR2 to {5, 7, 8, 9}, FR3 to {5, 13, 14, 15, 17}, and FR4 to {16, 19}.

The TFM of IS is illustrated in Fig. 5a. After formal identification of the IS, a subsystem of the BS, it excludes only two functional features - 21 and 22.

System goals to IS set by users are needed for TFM decomposition into use cases. The goals are stated as follows: SG1 “Register a reader”, SG2 “Check out a book”, SG3 “Return a book”, SG4 “Pay a fine”, SG5 “Impose a fine”, and SG6 “Close a fine”. All system goals are set by the librarian.

Tracing: The mappings from the TFM of IS to corresponding functional features of the TFM of BS are one-to-one. The mappings from system goals to the TFM of IS
are illustrated in Fig. 5b. Thus, SG1 “Register a reader” can be achieved by execution of functional features 2, 3, and 4; in turn, SG2 “Check out a book” – by 5, 6, 7, 8, 9 and 10, SG3 “Return a book” by 5, 13, 14, 15, 16, and 17, SG4 “Pay a fine” by 5, 18 and 19, SG5 “Impose a fine” by 16, and SG6 “Close a fine” by 19.

Fig. 5. The TFM of Information System (a) and mappings from system goals to it (b)

SG1 corresponds to FR1, SG2 to FR2, and SG3 to FR3. In turn, SG4, SG5 and SG6 together correspond to FR4. However, functionality specified by functional features 6, 10, and 18 belongs to the information system as such, but it is excluded from the software system, i.e., it will not be implemented and will remain manual. Besides that, functional features 1 and 12 indicate on the input data from the business system to the information system; and features 11 and 20 indicate on the output data that the information system provides for other activities of the business system and its external environment.

4.2 The CIM – Business Requirements for the System

*Domain Vocabulary and System Requirements*. A conceptual class diagram and a use case model, are represented in Fig. 6a. The classes are driven from the TFM of IS. They are ReaderAccount, ReaderCard, Reader, Request, BookCopy, BookFund, and Fine. The detailed description of class properties is skipped here.

The use case diagram and descriptions of use cases are also derived from the TFM of IS by using corresponding system goals as a decomposition criteria and mappings from functional requirements as criteria for determination of use case flows. The use cases are “Register reader”, “Identify reader”, “Check out book”, “Return book” and “Close fine”. SG5 “Impose fine” could be decomposed as use case “Impose fine” – an extending use case for “Return Book”. However, it is not necessary in our case. In turn, “Identify reader” is an inclusion use case for “Check out book”, “Return book”, and “Close fine”. All use cases are activated by an actor “Librarian”.

**Tracing:** Mappings from the classes and use cases to the TFM of IS are illustrated in Fig. 6b. As we can infer from these mappings, the use case “Register reader” operates with classes ReaderAccount, ReaderCard, and Reader. “Indentify reader” operates with Reader. “Check out book” operates with RequestForBook, BookCopy, and BookFund. “Return book” operates with BookCopy, BookFund, and Fine. Both “Impose fine” and “Close fine” operate with Fine. Thus, possible changes of classes
Reader, BookCopy, BookFund and Fine may have impact on several interrelated business processes and corresponding functional units.

![Diagram of conceptual class diagram and use case diagram (a), and their mappings to functional features of the TFM of IS (b)](image)

**Fig. 6.** The conceptual class diagram and use case diagram (a), and their mappings to functional features of the TFM of IS (b)

For each use case a system sequence diagram (or diagrams) should be created. These diagrams are based on cause-and-effect relations of the TFM of IS and user functional requirements. Main and alternative flows of use cases should also be defined in strong correspondence with the functional features. However, we omit them here due to page limitation.

### 4.3 Illustration of Traceability Links in case the Change Request Occurred

Let us consider a case when a perfective change request occurs, i.e., a request of system modification in a result of changes in user requirements.

Let’s assume that requirement “FR2: The system shall check out a book copy” is extended to “FR2: The system shall allow a reader to complete and register his/her request via Internet. The system shall perform check out a book copy by the registered request by both reader and librarian. The system shall inform a reader about the status of the registered request via e-mail.”

The trace links among elements before possible modification are the following: the user requirement FR2 is linked with TFM functional features 5, 7, 8, and 9 (Fig. 4b), which in their turn are linked with the system goal SG2 (Fig. 5b). If we consider these functional features within the TFM of IS (Fig. 5a), then we will be able to see that functionality touched by the modification (Fig. 7a) depends on functional features 2
and 3 (new reader registration), 6 (completing the request for a book) and 17 (taking the requested book copy from the book fund). Besides that, it affects functional features 10 (giving out the book copy to a reader), 13 (taking back the book copy from a reader) and 18 (paying the fine). This means that requested changes may generate side effects in these functional parts which origin from different user requirements and belong to different system goals (Fig. 7b) and use cases – “Identify reader” and “Check out book” (Fig. 6b). Hence, cause-effect relations between these functional features and modified/new functionality must be carefully verified.

By analyzing the TFM, we can see that completing of a request is manual. At the beginning, the librarian identifies a reader, then the reader is allowed to complete the request manually, and only after that the librarian registers the request. Thus, one part of the new requirement is already specified by feature 6. However, the new execution order not only saves this possibility, but also extends it by introducing a new path – the reader should provide his login data in order to be identified by the system, then he should be able to complete an electronic request form and register it in the system. Hence, features 5 and 7 must be modified; the new cause-and-effect relation from
feature 5 to feature 6 must be established (Fig. 7c). The next point is that identification of a reader via Internet will modify also the class Reader (Fig. 6b), e.g. by adding new class properties – login name, login password and e-mail, as well as functionality of the use case “Register reader” specified by features 2 and 3.

Another point is that a reader who has been registered his/her request via Internet should be notified about the status of the request. Therefore, new functional features “23: Updating the status of the request of a reader, {the request is registered OR the book copy is taken from the book fund}, System”, “24: Creating an e-mail, {}, System”, and “25: Sending the e-mail to a reader, {}, System” and cause-and-effect relations from features 7 and 8 to 23, from 23 to 24, and from 24 to 25 are established (Fig. 7c). Additionally, the new class E-mail also is created (Fig. 7e).

Besides that, giving out a book copy should also be modified, since a reader can came to the library to receive the book already taken from the fund. Checking out the book copy (feature 9) should also be related to the mentioned Internet functionality. Hence, a new feature is created – “26: Checking the status of the request of a reader, {}, Librarian” that generates feature 9 only if the book copy has been taken from the fund. A new alternative path in the TFM of IS is “5-26-9” (Fig. 7c).

Modifications of the TFM of IS must be propagated to the lower artifacts. As we speak in terms of use cases, this means that system goals should be verified. A new goal is stated by the reader – SG7 “Complete the request via Internet” that includes features 5, 6 and 7. In its turn, SG2 is extended and includes also features 23, 24, 25 and 26 (Fig. 7d). This modifies conceptual class and use case diagrams as shown in Fig. 7e in correspondence with new traceability links. A new use case “Complete request” is specified. It extends the use case “Check out book” and includes “Identify reader”.

Summarizing, implementation of changes in FR2 will require modifications in FR1 and FR3 that will cause creation of the new class E-mail and use case “Complete request” as well as modification of classes ReaderAccount, ReaderCard, Reader, Request and use cases “Register reader”, “Check out book” and “Identify reader”. Moreover, use cases “Return book” and “Close fine” may require re-testing.

Conclusion

MDA gives new breath to post-RS traceability support. However, automated support of pre-RS traceability requires formalization of requirements, business, knowledge models and relations among them. The TFM as a formal CIM-Business Model gives an opportunity to establish and use trace links between elements of development artifacts for pre-RS traceability needs. This results in feasibility to trace system requirements to and verify them with business functionality and original user requirements, check necessity of modification of logically related parts, and propagate established modifications in business models to system requirements and classes.

Direction of our on-going and further research is formalization of transformations between CIM models and from CIM to PIM by means of topological functioning modeling and “lightweight” mathematics.
Erika Asnina, Bernards Gulbis, Janis Osis, Gundars Alksnis, Uldis Donins, Armands Slihte

References


