RIGA TECHNICAL UNIVERSITY
Faculty of Computer Science and Information Technology
Institute of Applied Computer Systems

Armands ŠLIHTE
Student of the Doctoral Study Program "Computer Systems"

THE INTEGRATED DOMAIN MODELING: AN APPROACH & TOOLSET FOR ACQUIRING A TOPOLOGICAL FUNCTIONING MODEL

Doctoral Thesis

Scientific Supervisor
Dr.habil.sc.ing., Professor
J. OSIS

Riga 2015

ANOTĀCIJA

Priekšmetiskās vides modelis un priekšmetiskās vides modelēšana ir svarīga programmatūras inženierijas, kā arī priekšmetiskās vides analīzes sastāvdaļa. Pastāv daudzas pieejas, lai modelētu priekšmetisko vidi, taču tās parasti balstās uz "labo praksi", nevis uz formālu priekšmetiskās vides modeli. Tā ir būtiska problēma, bet programmatūras inženierijā tas jo īpaši noved pie tā, ka nepastāv formālas saiknes starp priekšmetiskās vides modeli un risinājuma modeli. Tādējādi neveidojas formāla saite arī starp priekšmetisko vidi un risinājumu, kas noved pie neveiksmīgiem programmatūras izstrādes projektiem, pārtērētiem budžetiem un/vai sliktas kvalitātes programmatūras. Izvairoties no formālas priekšmetiskās vides analīze programmatūras inženierijas sākumā nav iespējams veikt priekšmetiskās vides modeļa satura un darbības apgabala validāciju, kā arī nav iespējams veikt formālu transformāciju no priekšmetiskās vides modeļa uz risinājuma modeli, un līdz ar to nav iespējams formāli atsekot priekšmetiskās vides modeļa elementiem risinājuma modelī un otrādi.

Promocijas darba “Integrētā priekšmetiskās vides modelēšana: pieeja un rīku kopa topoloģiskā funkcionēšanas modeļa iegūšanai” galvenais mērķis ir uzlabot priekšmetiskās vides analīzes procesu, kā arī samazināt programmatūras izstrādes projektu izmaksas, pagaustinot programmatūras kvalitāti un uzlabojot izpratni par priekšmetisko vidi programmatūras inženierijas procesā.

Lai sasniegtu izvirzīto mērķi autors ir izstrādājis jaunu priekšmetiskās vides modelēšanas pieeju un atbalstošo rīku kopu, ar kuras palīdzību iespējam iegūt matemātiski formālu priekšmetiskās vides modeli, kas ir transformējams un kas var tikt izmantots kā no skaitļošanas neatkarīgais modelis modeļu vadāmajā arhitektūrā (MDA). Šī pieeja balstās uz topoloģisko funkcionēšanas modeli (TFM), ontoloģiju un lietošanas gadījumiem, kā arī uz dabīgās valodas apstrādi. Atbalstošā rīku kopa balstās uz MDA standartiem, un nodrošina modeļu transformāciju, lai automātiski iegūtu sākotnējo TFM no formāli definētām priekšmetiskās vides zināšanām (deklaratīvajām un procedurālajām zināšanām).

Promocijas darbs sastāv no 224 lappusēm, 52 attēliem, 4 tabulām un 10 pielikumiem. Literatūras sarakstā iekļauti 100 literatūras avoti.

ABSTRACT

Domain model and domain modeling is an important part of software engineering and domain analysis in general. There are many domain modeling approaches available and being used today, however these tend to be based on "best practices" and not based on a formal domain model. This is an essential problem, but in particular for software engineering this leads to a lack of formal connection between the domain model and the solution model. In order this leads to a gap between the domain and the solution, thus resulting in failed software development projects, overrun budgets and/or bad quality software. By avoiding a proper domain analysis at the beginning of software engineering with a formal domain model it is not possible to perform scope and content validation of the domain model, it is not possible to do a formal transformation from domain model to the solution model, and thus it is not possible to formally trace the elements of the domain model to the solution model.

The main goal of this doctoral thesis titled "*The Integrated Domain Modeling: An Approach & Toolset for Acquiring a Topological Functioning Model*" is to improve the process of domain analysis and sequentially lowering cost of software development projects, raising the quality of produced software and enabling their success by improving the understanding of the domain at the beginning of software engineering process.

To achieve this goal a novel domain modeling approach and a supporting toolset has been developed by the author for acquiring a mathematically formal domain model that is transformable and can be used as the Computation Independent Model (CIM) within Model Driven Architecture (MDA). This approach is called "*Integrated Domain Modeling*" (IDM) and is based on the Topological Functioning Model (TFM), Ontology and Use Cases. This approach provides a way to use directly use Ontology as input for software engineering by exploiting Use Cases and Natural Language Processing. The supporting toolset is based on MDA standards and provides a model-to-model transformation for acquiring an initial TFM automatically from formally defined domain knowledge (declarative and procedural knowledge).

The doctoral thesis consists of 224 pages, 52 figures, 4 tables, and 10 appendices. Bibliography includes 100 literary sources.

TABLE OF CONTENTS

INTRODUCTION

In the context of software engineering, a domain is most often understood as an application area, a field for which software systems are developed [74]. A domain model can be used as input for implementation of the solution within a software development process. The model elements that compose the domain model can serve as basis for code construction, which can be done manually or by using automated code generation as suggested by Model Driven Architecture (MDA) [41]. Domain model is an integrated system of models that reflect the enterprise, where software is to be applied [89]. In other words, domain model is the AS-IS model of the business organization and processes. Moreover, domain modeling is a human activity that leads to the creation of different types of domain representations. These representations may be tacit (in human minds) and explicit/externalized (on paper or in a software tool) [36]. Furthermore, domain analysis is a process where the necessary information for developing software systems for a specific domain is identified, captured, structured, and organized for further reuse [74]. According to [58], there are many domain-modeling approaches, but these are mostly based on unification of standards rather than mathematically formal models, with exception of Petri Nets and Topological Functioning Model (TFM). This is a serious issue for software engineering, because there is no formal connection between the domain model and the solution.

Motivation of the Research

In his research Capers Jones [30] analyzes positive and negative innovations in software engineering, and concludes that the way software is built remains surprisingly primitive. His research [30], [96] has found that majority of software development projects fail because of overrun budget and schedule, or have hazardously bad quality level of the produced software. Some of the issues (which have not changed during 30 years) mentioned in the research are as follows: “Initial requirements are seldom more than 50% complete”, “There are more defects in requirements and design than in source code”, “Finding and fixing bugs is the most expensive software activity”. The main problem is that software development is not a well-structured discipline, requires

high content of manual labor and the processes are not automated. The author of this thesis believes that a formal domain model is the key to software development automation, and that MDA is a positive innovation for software development. However, inability to produce a formal domain model within software engineering leads to the following problems. The scope and content of the acquired domain model cannot be validated in an automated way (only manual, subjective validation is possible). It is not possible to transform the domain model to the solution model automatically (again only manual, subjective transformation is possible). Thus there is no formal transformation, and it can also be problematic to formally trace the elements of the domain model to the solution model (after several iterations of manual transformation elements can be lost or incorrectly transformed). This leads to low quality of the software and an inefficient way of producing it. The ultimate goal of this research is to lower the cost and raise the quality of software development by introducing a methodology and a toolset, which would allow a comprehensive analysis of the domain in the beginning of software development based on a formal domain model; thus minimizing the number of bugs and change requests due to an inconsistent understanding of the domain.

Research Area

Model Driven Architecture (MDA) proposes software development to abstract from the code as the uppermost of the functionality of the information system to the model of the information system [23]. MDA is a software development framework, which defines 3 layers of abstraction for system analysis: Computation Independent Model (CIM), Platform Independent Model (PIM), and Platform Specific Model (PSM). CIM describes system requirements and the way system works within its environment while details of the application structure and realization are hidden or are yet undetermined. CIM is also known as the domain model or the business model. As the business model CIM should be a precise description of the business in its environment, by the business, in the language of business people, dedicated to business purposes [5]. However, the only formal means to define a CIM is Petri nets, which are too complex for business people since it is based on “heavy” mathematics

[5]. Another formal model that can be used as CIM is the Topological Functioning Model (TFM).

This work is a part of the Topological Functioning Model for Software Engineering (TFM4SE) research. TFM is a domain model, which offers a formal way to define a system by describing both the system's functional and topological features [4]. TFM represents the system in its business environment and shows how the system is functioning, without details about how the system is constructed. This research suggests using a TFM as the Computation Independent Model (CIM) same as Topological Functioning Model for Model Driven Architecture (TFM4MDA) approach [53], [64], [4], [55] and [61]; acquiring a mathematically formal and thus transformable CIM. In related research [12] the TopUML approach is described for software development with emphasis on topology, where Platform Independent Model/Platform Specific Model (PIM/PSM) is supplemented with topology. TopUML is a UML profile and approach for introducing cause and effect relationships into the UML based on the topology of TFM. TopUML approach suggests sequential phases of TFM4MDA approach to be combined for fulfilling the Model Drive Architecture (MDA) life cycle taking the TFM as source for PIM/PSM. Although the TFM, TFM4MDA and TopUML provide a solid basis for CIM construction within MDA and further transformations to PIM/PSM, until now the construction of the TFM relies on a heavy manual process with no tool support and a poor integration with the common IT practices. The AS-IS processes of TFM4MDA are described in [56], [58], [60] and for TopUML in [13].

Purpose of the Research

The goal of the thesis is to improve the process of domain analysis by providing an approach and a supporting toolset for acquiring a formal domain model that is transformable and can be used as CIM within MDA. Thus lowering cost of software development projects, raising the quality of produced software and enabling their success by improving the understanding of the domain.

The tasks of the thesis in order to achieve the goal are defined as follows:

1. Analyze the existing domain modeling approaches to identify their strengths and weaknesses, and conformance to CIM within MDA;
2. Evaluate the domain modeling approaches based on their formality, conformance to MDA and practical usability;
3. Analyze the TFM approach to identify its strengths and weaknesses and points for improvement;
4. Develop the Integrated Domain Modeling (IDM) approach for acquiring a formal domain model in the form of TFM based on formal knowledge about the domain using existing IT pratices;
5. Develop a toolset to support the IDM approach based on MDA standards, which consists of Use Cases Editor, TFM Editor, and Use Cases to TFM Transformation tool;
6. Conduct a case study using the IDM approach and toolset for a real software development project;

The research subject is domain modeling with focus on the domain model within software engineering.

The research objects are MDA and TFM, focusing on how to acquire a CIM in a formal way with accordance to MDA standards.

The research methods used – analysis by comparison, metamodeling method and model transformation, design science, and case study.

Thesis statements are the following: 1) If for a particular business domain corresponding domain knowledge is formally defined, then it should be possible to generate a formal domain model automatically; 2) The usability of TFM can be significantly improved by addressing the issues of informal description and lack of tool support with a new approach based on common practices; 3) Declarative and procedural knowledge complement each other and enable a thorough domain analysis.

Scientific Innovation and Practical Value

The **scientific innovation** of this research is a novel approach called Integrated Domain Modeling (IDM) for domain modeling, which provides means to acquire a mathematically formal domain model in the form of TFM, and at the same time uses

common standards as key input for the domain modeling process and introduces a model transformation. The IDM is based on the declarative an procedural aspects of domain knowledge using Ontology and Use Cases as input, and executing a model transformation to TFM using also Natural Language Processing (NLP).

The **practical value** of this research is the supporting toolset for IDM approach that consists of the Use Cases Editor, TFM Editor and Use Cases to TFM Transformation tools. As well as, a case study of applying the IDM approach for e-commerce software development project. Before there were no tools to support the TFM approaches, but now with the IDM approach it is possible to acquire a TFM with an automatic model transformation.

Artifacts created during this research are as follows:

- IDM Use Cases metamodel according to Meta-Object Facility (MOF);
- IDM TFM metamodel according to MOF;
- Use Cases to TFM model transformation according to Query/View/Transformation Operational (QVTo);
- Use Cases Editor tool based on Eclipse Modeling Framework (EMF);
- TFM Editor tool based on Eclipse EMF;
- TFM Diagram Editor tool based on Eclipse Graphical Modeling Framework (GMF);
- Use Cases to TFM Transformation tool based on Eclipse Plugin Development Environment (PDE);
- Case Study for e-commerce software development project using the IDM approach and toolset.

Approbation of the Work Results

The main results of the research are presented in the following 8 international scientific conferences (4 were held in Latvia and 4 in foreign countries):

1. 11th International Baltic Conference on DB and IS (DBIS 2014), Estonia, Tallinn, June 8-11, 2014;
2. 54st Scientific Conference of Riga Technical University, Riga, Latvia, October, 2013;

3. 7th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2012), Poland, Wrocław, June 29-30, 2012;
4. 3rd International Workshop on Model-Driven Architecture and Modeling-Driven Software Development (MDA & MDSD 2011), China, Beijing, June 8-11, 2011;
5. 9th International Baltic Conference on DB and IS (DBIS 2010), Latvia, Riga, July 5-7, 2010;
6. 2nd International Workshop on Model-Driven Architecture and Modeling Theory-Driven Development (MDA & MTDD 2010), Greece, Athens, 2010;
7. 51st Scientific Conference of Riga Technical University, Riga, Latvia, October, 2010;
8. 13th East-European Conference (ADBIS 2009), Latvia, Riga, September 7-10, 2009;

The main results of the research are published in the following 11 scientific papers:

1. Šlihte A. Introduction to Integrated Domain Modeling Toolset// Scientific Journal of RTU. Computer Science. - 2014. (to be published)
2. Šlihte A. The Integrated Domain Modeling: A Case Study// In Proceedings of the 11th International Baltic Conference, Baltic DB&IS 2014. TUT Press, 2014. - pp. 465-470. [ISBN 978-9949-23-633-6]
3. Osis J., Šlihte A., Jansone A. Using Use Cases for Domain Modeling// Proceedings of the 7th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2012). Poland, Wrocław, June 29-30, 2012. Lisbon: SciTePress, 2012. – pp. 224-231. [ISBN 9789898565136, Indexed by Thomson Reuters, Inspec, EI, DBLP]
4. Doniņš U., Osis J., Šlihte A., Asņina Ē., Gulbis B. Towards the Refinement of Topological Class Diagram as a Platform Independent Model// Proceedings of the 3rd International Workshop on Model-Driven Architecture and Modeling-Driven Software Development (MDA & MDSD 2011). China, Beijing, June 8-11, 2011. Lisbon: SciTePress, 2011. – pp. 79-

88. [ISBN 9789898425591, Indexed by Thomson Reuters, Inspec, EI, DBLP, ISTP]

5. Šlihte A., Osis J., Doniņš U. Knowledge Integration for Domain Modeling// Proceedings of the 3rd International Workshop on Model-Driven Architecture and Modeling-Driven Software Development (MDA & MDSD 2011). China, Beijing, June 8-11, 2011. Lisbon: SciTePress, 2011. – pp. 46-56. [ISBN 9789898425591, Indexed by Thomson Reuters, Inspec, EI, DBLP, ISTP]
6. Šlihte A., Osis J., Doniņš U., Asņina Ē., Gulbis B. Advancements of the Topological Functioning Model for Model Driven Architecture Approach// Proceedings of the 3rd International Workshop on Model-Driven Architecture and Modeling-Driven Software Development (MDA & MDSD 2011). China, Beijing, June 8-11, 2011. Lisbon: SciTePress, 2011. – pp. 91-100. [ISBN 9789898425591, Indexed by Thomson Reuters, Inspec, EI, DBLP, ISTP]
7. Asņina Ē., Gulbis B., Osis J., Alksnis G., Doniņš U., Šlihte A. Backward Requirements Traceability within the Topology-based Model Driven Software Development// Proceedings of the 3rd International Workshop on Model-Driven Architecture and Modeling-Driven Software Development (MDA & MDSD 2011). China, Beijing, June 8-11, 2011. Lisbon: SciTePress, 2011. – pp. 36-45. [ISBN 9789898425591, Indexed by Thomson Reuters, Inspec, EI, DBLP, ISTP]
8. Šlihte A. The Concept of a Topological Functioning Model Construction Tool// Advances in Databases and Information Systems: 13th East-European Conference, ADBIS 2009. Associated Workshops and Doctoral Consortium, Local Proceedings, Latvia, Riga, September 7-10, 2009. - pp. 476-484.
9. Šlihte A. The Specific Text Analysis Tasks at the Beginning of MDA Life Cycle// Databases and Information Systems Doctoral Consortium, Latvia, Riga, July 5-7, 2010. – pp. 11-22.
10. Osis, J., Šlihte, A. Transforming Textual Use Cases to a Computation Independent Model// Model-Driven Architecture and Modeling Theory-Driven Development: Proceedings of the 2nd International Workshop on

Model-Driven Architecture and Modeling Theory-Driven Development (MDA & MTDD 2010). Greece, Athens, July 22-24., 2010. Lisbon: SciTePress, 2010. - pp. 33-42. [ISBN 9789898425164, Indexed by SCOPUS, Thomson Reuters, Inspec, DBLP]

11. Šlihte A. Implementing a Topological Functioning Model Tool// Scientific Journal of RTU. 5. series., Computer Science. - 43. vol. - 2010. - pp. 68-75.

The resulting IDM approach and toolset were used for a real software development project done by Pearl Consulting AS, Norway (SIA "Pearl Baltic Labs" in Riga, Latvia) [71] for a customer who is in subscription commerce business, which operates in Norway, Sweden, Denmark, Finland and United Kingdom. The project outcome was an e-commerce system for the subscription commerce business.

Outline of the Thesis

The thesis includes an introduction, 4 sections, conclusions, 10 appendices, and bibliography. The doctoral thesis consists of 216 pages, 52 figures, and 4 tables. Bibliography includes 92 sources.

Introduction explains the motivation of the thesis, defines the research goal and the tasks to achieve the goal, novelty and practical value of the research, approbation and the main results.

Section 1 gives a definition of a domain model and domain modeling, analyzes some of domain modeling approaches to give an overview of the different approaches available. Since the goal of this thesis is to improve domain analysis the Model Driven Architecture (MDA) is analyzed to introduce common standards and model transformation possibilities for a domain model. Domain modeling approaches are evaluated based on three sets of criteria – formality of domain model, conformance to MDA, and practical usability. The problems of existing approaches are analyzed.

Based on the analysis of domain modeling approaches *Section 2* first defines a solid basis for the Integrated Domain Modeling (IDM) approach developed by the author by analyzing the strength and weaknesses of the Topological Functioning Model (TFM) approaches. Then the innovation and basic principles of the IDM approach are defined including knowledge integration, natural language processing,

and generating a Computation Independent Model (CIM) from formally defined knowledge with a model-to-model transformation. The author used a library business system as an example to demonstrate the IDM approach.

Section 3 introduces the supporting toolset for the IDM approach developed by the author, discussing the scope, architecture, used technologies, implementation, model transformation, and application. This toolset is based on Eclipse [17] and consists of several tools that interoperate to provide a possibility for a particular domain model construction.

Section 4 provides a cases study of applying the IDM approach and the supporting toolset to a real software development project for a Subscription Commerce Business (SCB) in the area of e-commerce. SCB operates in Scandinavia and United Kingdom and their e-commerce business consists of these main processes – article management, bundle management, product management, customer subscription, bundle delivery, and batch purchase. These processes are modeled with the IDM toolset defining the Ontology and Use Cases, and acquiring a corresponding TFM with a model-to-model transformation.

Conclusions summarize the results of this doctoral thesis - results of analyzing the existing domain modeling approaches, results of analyzing strengths, weaknesses and points for improvement of the TFM approach, the IDM approach, the IDM toolset, and conclusions for the conducted case study. Possible future research directions are also defined.

Thesis contains ten *appendices*:

1. List of figures;
2. List of tables;
3. IDM Use Cases Metamodel According to Ecore;
4. IDM TFM Metamodel According to Ecore;
5. IDM Toolset’s Use Cases Editor Artifacts;
6. IDM Toolset’s TFM Editor Artifacts;
7. IDM Toolset’s TFM Diagram Tool Artifacts;
8. IDM Toolset’s Use Cases to TFM Transformation Tool Artifacts;
9. IDM Toolset User Guide;
10. Survey On the IDM Toolset.

1. DOMAIN MODELING AND MODEL DRIVEN ARCHITECTURE

This section gives an overview for the topic of domain modeling and analyzes some of the existing approaches, also discussing the Model Driven Architecture (MDA) as the umbrella standard for modeling in general. Author first gives the definition and purpose of domain modeling, then reviews literature on the different approaches available and analyzes their strengths and weaknesses. The importance of model and modeling approach conformity with MDA is stressed.

1.1. Domain Modeling

Domain model and domain modeling is a very important part of software engineering, however these terms have been used in different contexts having different meanings. For example, in object-oriented programming a class diagram is sometimes called a domain model. However, for this research the author is going to use the following definition of a domain model. Domain model is a representation of a particular problem domain or business domain from a "computation independent" perspective that exists or can exist in real life, which includes terminology, concepts, relationships, rules, and business processes. Moreover, domain model has to be built by the business people (it can be done with the help of IT experts) and it has to be understood by the business people (although after the domain model is built and approved for a particular purpose it might directly be used only by IT experts and/or machine). The author will first discuss the purpose of domain modeling and then give the different definitions of a domain model from 3 perspectives – viewpoint, scope and present (up-to-date).

1.1.1. Purpose of a Domain Model

First the author discusses the purpose of domain model and modeling. In MDA guide [43] the OMG points out that if we built buildings the way we build software,

we would be unable to connect them, change them or even redecorate them easily to fit new uses; and worse, they would constantly be falling down. A proper designing phase and a domain model is necessary to save development effort, since there are less "bugs" or inconsistencies with what was expected of the software. Another reason is to integrate and maintain the produced software after the implementation phase. Without a proper documentation for the solution (including the domain model) it is hard for new experts (people who have not worked on the first implementation) coming into the project to develop integration with this solution or do maintenance work. Moreover, even if the same experts do the work it is vital to the success of the project in long term to have a good knowledge base and structure of the documentation for the solution. OMG mentions another benefit for having a formal domain model, i.e., it is possible to automate at least some of the construction of the software solution [43]. Software development projects do not have a positive historical track record according to [30]. Capers Jones analysis shows that majority of software development projects fail because of overrun budgets and schedules, or hazardously bad quality levels. Software development is not a well-structured discipline and the processes are not automated in any way, as oppose to traditional engineering, like the construction example given by OMG. The authors in [56] state that the domain model should be the cornerstone of software development – it is a design, documentation and a way of decreasing inconsistencies and over budget costs.

Another purpose of domain modeling that is rarely used separate from software engineering is domain analysis. Usually when stakeholders talk about domain analysis this is in context of a software solution that could improve the business process by saving money for the company or earning more money with better service to their customers. However, in author's opinion domain analysis can be treated separately from software engineering, since theoretically a company would still benefit of modeling their domain and doing domain analysis without any software solutions. Of course, it is hard to imagine a company without software solutions today. Nevertheless, there could be no software support for part of the business processes in a problem domain. A domain model can provide useful insights for the company of their business processes and structure.

1.1.2. Viewpoint of a Domain Model

Viewpoint describes what should be modeled within a domain model and what should not. In terms of software engineering it can be AS-IS and/or TO-BE model of real world business system in its environment. It is important to note here, that domain model is not meant for software requirements, but instead should show the requirements or the functioning of the actual business system (AS-IS and/or TO-BE). The term computation independence comes from Model Driven Architecture's (MDA, discussed in detail in section 1.3) definition of Computation Independent Model (CIM). CIM means a model that does not show exactly information processing by digital computation means. CIM may reflect computation and logic that must be executed by a computer (information system or software), but specification of a type of the computer or how the computer organizes or performs this is forbidden [5]. Thus, for a computation independent perspective one's focus lies on real life business processes and concepts as oppose to software routines and structures. Domain modeling should show how a business system works within its environment, while the details of the application structure and realization should be hidden or not yet determined. Domain model is independent from "computation" and it can live a life on it's own – without information systems, software engineering and software development. Businesses may choose to use domain modeling to analyze their business organization, to improve their business processes without additional software, or to improve their business governance and internal documentation. However, for software engineering a domain model is the link to the real life business system and concepts. Requirements engineering is an essential part of software and systems development. Besides the elicitation, analysis, and specification of the intrinsic system requirements as a basis for these activities, it also involves the elicitation, analysis, and specification of the information about the domain [7]. Thus, also in requirements engineering a set of requirements are computation independent or at the domain level (as oppose to the solution level).

1.1.3. Scope of a Domain Model

Scope describes how to define the borders of a domain model. In theory there could be a domain model of everything in the world, but in practice of course this is not possible. In practice a domain model has its scope and focuses on a particular problem and aspects of the problem. Here is a definition of the domain model from author of [7]: "*A domain model gathers all the information about a domain as needed to understand and formulate the requirements on a particular application system to be developed in software and systems engineering. Domain knowledge and domain properties, in contrast to requirements, cannot be chosen freely, but have to reflect facts about the application domain and the operational context of the System or Software under Construction. This is in contrast to requirements that can be chosen freely according to the stakeholder needs.*" This is a good definition since it shows the difference between the domain model and requirements. Requirements are a more focused part of the domain model.

In the context of MDA, CIM is the domain model and constructing a CIM is domain modeling, however the domain model can have a bigger scope than CIM (this is discussed in section 1.4). According to [5] CIM may include 3 main parts: 1) CIM-Knowledge Model; 2) CIM-Business Model; 3) CIM-Business Requirements. CIM-Knowledge Model representation is mostly unstructured natural language – interviews, work instructions, process descriptions, or semi-formal languages such that business managers or domain experts are able to use. CIM-Business Model and CIM-Business Requirements include more or less structured language models accepted by requirements community, e.g. the Language Extended Lexicon (LEL) and scenarios, Semantics for Business Rules and Business Vocabulary, and Use Cases. Another means for CIM definitions are different languages and notations for business process modeling that may include semi-formal languages, such as ARIS-Event Process Chains, Business Process Modeling Notation (BPMN) used in Business Process Modeling (BPM), Vide CIM Level Language (VCLL), UML profiles, UML Activity Diagrams [5]. Moreover, in domain modeling different forms of Ontologies are developed to capture and document domain knowledge. Author of [7] explain that to capture domain models quite different kinds of modeling techniques have to be used to

represent the different parts of domain models. And if for a given domain no comprehensive domain models exist yet, it is necessary to work out an adequate domain theory during software and system development. Thus, within the scope of a domain model there are different aspects that can be modeled. Some of these domain-modeling approaches are going to be discussed further on in this section.

Identification of scope for a domain model is not an easy task. Some of the approaches provide special techniques for doing this, and some do not. One example of a formal way to identify the scope is provided by the Topological Functioning Model (TFM) approach, which will be discussed further on.

1.1.4. Actuality of a Domain Model

Another important perspective for a domain model is whether it is present or up-to-date. Since the real world is constantly changing any domain model is a snapshot of the particular moment in time it has been produced. Research [35] stresses the context and time dependency of the CIM. Authors explain that the context for an organization modeling their domain can be very different (e.g. no software exists, additional software required, reconfiguration of existing software required) and the actual domain (researchers call it human intelligence model or the human knowledge about the domain) is changing based on time. This is a really important point for domain modeling. The fact that none of existing approaches transparently considers information processing by human and artificial intelligence suggests that new methods are to be developed for modeling at CIM level [35]. The author agrees with this research, however the lack of time dimension and distinction between human and machine knowledge of the CIM does not destroy its importance in software engineering and domain analysis in general. Moreover, in author's opinion, updating CIM on a regular basis should be common practice (thus the business policy should be – if something does not exist in CIM it is not considered as part of domain, but an ad-hoc solution and should be considered for formal inclusion into CIM) and easy to use approaches need to be provided to the business people and system analysts to do so. Authors of [5] note that the only formal means for definition of a domain model are

Petri nets and their extensions, and also TFM. CIM should be formally defined, thus it should be understandable by a machine.

1.2. Domain Modeling Approaches

This sub-section gives an overview of the currently used domain modeling approaches and positions the TFM among them. The author based on his 10-year experience as a software engineer/researcher has chosen a set of approaches to represent domain modeling from different perspectives. Nevertheless, these are just a few approaches and this is meant to be an overview of domain modeling trends rather than a comprehensive study on domain modeling approaches. The approaches that are going to be described here are these – Business Process Modeling Notation (BPMN), Event-Driven Process Chains (EPC), Ontology and Use Cases, Linguistic Assistant for Domain Analysis (LIDA) [76], Natural Language Requirements Analysis (NIBA in German) [72] and Topological Functioning Model (TFM). A more detailed explanation of the choice of approaches follows below.

Business Process Modeling and Notation (BPMN) approach was chosen for its wide acceptance in the software industry and a good tool support from many vendors. BPMN is now basically the industry standard for domain modeling. One of positive innovators mentioned by Capers Jones [30] is SAP [77]. SAP use Event-Driven Process Chains (EPC) to model processes supported by their ERP and other products. Ontology was chosen for its strength to describe the structure of the domain and also the its formal nature – it can be understood by machine. Although Use Cases are not standardized, they are the most popular means for the practitioners to define the domain. Some research suggests also UML Activity Diagram can be used for business process modeling and not only software modeling (solution model vs. domain model). Thus this approach might be considered for domain modeling as well since UML is widely accepted, but is not going to be discussed here. On the other hand, Petri Nets is a mathematically formal model, but it is usually hidden from users of business models. The cause is both hard mathematics and representation of domain information that is not intuitive for users [5]. For these reasons also Petri Nets will not be discussed. It is important for the business people to understand it, because they are the main

stakeholders for the domain model and it's their responsibility to validate and approve it. Otherwise there is a big risk for the domain model to be inconsistent with the actual business, which in worst-case scenario might lead to a failed software development project or a failed business.

Some of innovative and interesting domain-modeling approaches are also reviewed to show some potential trends in domain modeling approaches, which exploit artificial intelligence – LIDA [76] and NIBA [72]. There have actually been other interesting attempts to acquire a domain model worth mentioning. Approach proposed in [22] suggests generating implementation from textual Use Cases. This approach uses statistical parser on Use Cases and by analyzing the parse trees compose so called Procases for further use in implementation generation. Procases can be thought of as a formal model of requirements. Another approach ReDSeeDs [32] defines software cases to support reuse of software development artifacts and code in a model driven development context. This approach is very complex and it depends on writing the software cases very precisely by adding specific meaning to every word or phrase of software case sentences. The Use Case Driven Development Assistant (UCDA) tool's methodology follows the IBM Rational Unified Process (RUP) approach to automate the class model generation [87]. It starts with analyzing the requests of stakeholders and identifies actors and Use Cases. From there the tool can generate the system's Use Case diagram, class diagram, collaboration diagram, and other artifacts. The tool uses natural language parsing to achieve this. This methodology deals only with identifying Use Cases, but not how they operate. The steps of the main scenario or the basic flow of events have to be defined manually.

1.2.1. Business Process Modeling Notation

The Business Process Modeling and Notation (BPMN) was developed in 2002 by Stephen A. White, an IBM employee, and made public by Business Process Management Initiative (BPMI). In 2005, BPMN was transferred from BPMI to Object Management Group (OMG) [97]. BPMN is a standard maintained by the OMG since 2006 [51]. The primary goal of BPMN is to provide a notation that is readily understandable by all business users, from the business analysts that create the initial

drafts of the processes, to the technical developers responsible for implementing the technology that will perform those processes, and finally, to the business people who will manage and monitor those processes [8]. BPMN provides the opportunity to the business people to analyze their business processes with a graphical notation. OMG claims that BPMN is simple enough for the business people to be able to understand it. At the same time BPMN is detailed enough to also serve as basis for implementation of the business process. BPMN allows modeling 3 different, but related aspects of a business domain – process, collaboration and choreography [3]. Process aspect represents a sequence of activities that constitute a business process. Collaboration aspect represents a process that has at least 2 participants. Choreography aspect represents a sequence of interactions between participants. BPMN has a relatively simple high-level structure, however complexity arises due to the many different type of available elements [3]. BPMN has 4 types of events (including different triggers an throw/catch semantics), 4 types of activity (including different variants), 5 types of gateway, 3 connecting objects, 2 artifacts, 4 types of data objects, etc.

BPMN has the 4 basic element categories: 1) Flow objects; 2) Connecting objects; 3) Swimlanes; 4) Artifacts [91]. Flow object category consists of events, activities and gateways. Event represents something that happens during a business process (e.g. event “Goods to ship” in Figure 1.1). Activity represents a generic work that a company performs (e.g. activity “Fill in a postal label” in Figure 1.1). Gateway represents a divergence and convergence of a sequence flow (e.g. gateway “Mode of delivery” in Figure 1.1). Connecting object category consists of sequence flow, message flow and association. Sequence flow represents order in which the activities will be performed in a process. Message flow represents flow of messages between 2 separate process participants. Association represents a relation between flow objects and artifacts, e.g., data or text. Swimlanes category consists of pool and lane elements. Pool represents a participant in a process (e.g. pool “Hardware retailer” in Figure 1.1). Lane represents a sub-partition within a pool (e.g. lane “Clerk” in Figure 1.1). Artifacts category consists of data object, group and annotation. Data objects are a mechanism to show how data is required or produced by an activity. Group can be used for documentation or analysis purposes but does not affect the sequence flow. Annotation is a mechanism to add comments to the diagram.

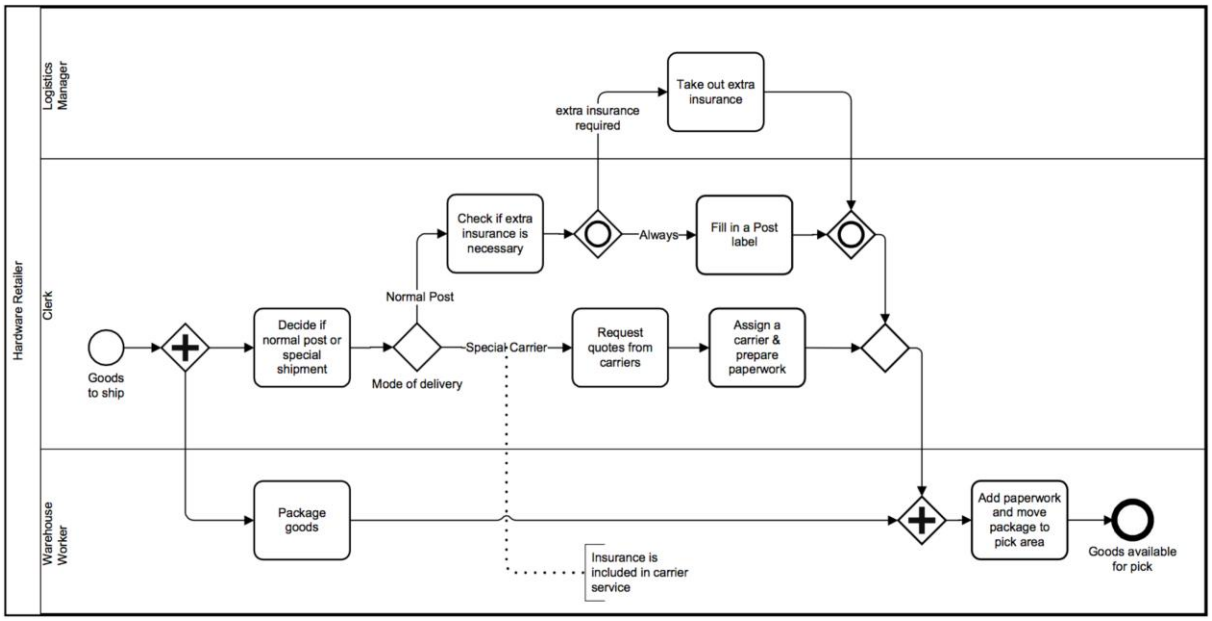


Figure 1.1. Example BPMN for Shipment Process (taken from [8])

Figure 1.1 shows an example BPM for steps a hardware retailer has to fulfill before the ordered goods can actually be shipped to the customer [8]. In this example, one pool and different lanes for the people involved are used in this process. BPMN can be used to define AS-IS, as well as TO-BE behavior of a business domain. BPMN provides a well-structured approach for domain modeling and has been based on the best practices of other approaches. There are also a lot of software tools to support the BPMN, e.g., Sparx Systems Enterprise Architect, MagicDraw, IBM Rational System Architect, ARIS, etc.

However, there are also some drawbacks to BPMN. First of all, the notation has quite a lot of element types, so the diagram can become very complex. This means that the business people as well as the technical experts need to have special training to learn BPMN to understand it. Organizational structure is indirectly represented in BPMN by pools and lanes. Also there is a special element Business Rule Task, which provides a mechanism for the process to provide input to a Business Rules Engine and to get the output of calculations that the Business Rules Engine might provide [8]. However, from the perspective of the domain modeling, BPMN does not include modeling organizational structure and business rules; this is also no the purpose of this standard (other standards need to be applied to define these aspects of a domain).

1.2.2. Event-Driven Process Chains

The event-driven process chain (EPC) was developed by the team of August-Wilhelm Scheer at the Saarbrücken University in a joint research project with SAP AG [77]. The EPC is now widespread in practical use especially in the German-speaking environment [97]. EPC is the key component of SAP R/3's [77] modeling concepts for business engineering and customization and has also been integrated in SAP's NetWeaver System. It is the modeling notation supported by the ARIS Process Platform [2], which provides an integrated toolset for designing, implementing, and controlling business processes [78].

EPC is represented by an ordered graph, which consists of events and functions, providing various connectors that allow alternative and parallel execution of processes [93]. Events define a circumstance under which a function or a process executes or which state it results in, e.g., "employee wants apply for a business trip" (see Figure 1.2). Functions define tasks or activities from an initial state to a resulting state within a business process, e.g., "fill out application". In case of possible different resulting states it is possible to use logical connectors making it a decision function. Process owner defines the responsible for a function, e.g., "Employee" and "Manager". Organization unit defines which organization within the structure of a business is responsible for a specific function, e.g. "Accounting department". Information, material, and resource objects define real world objects that can serve as input or output for a function, e.g., "Purchase order". Logical connectors enable the logical relationships between events and functions. Thus the control flow can be split and later synchronized. Logical relationships can be XOR, AND, or OR. To branch or merge control flow XOR is used, which activates one of the paths. To fork or join control flow the AND is used, which activates all paths concurrently. And OR relationship is used to activate one or more paths in the control flow. Control flow connects events with functions, process paths, or logical connectors. Information flow defines connection between functions and input or output data. Organization unit assignment defines the connection between an organization unit and the function. Process path defines connection from or to other processes and serve as navigation aid in the EPC.

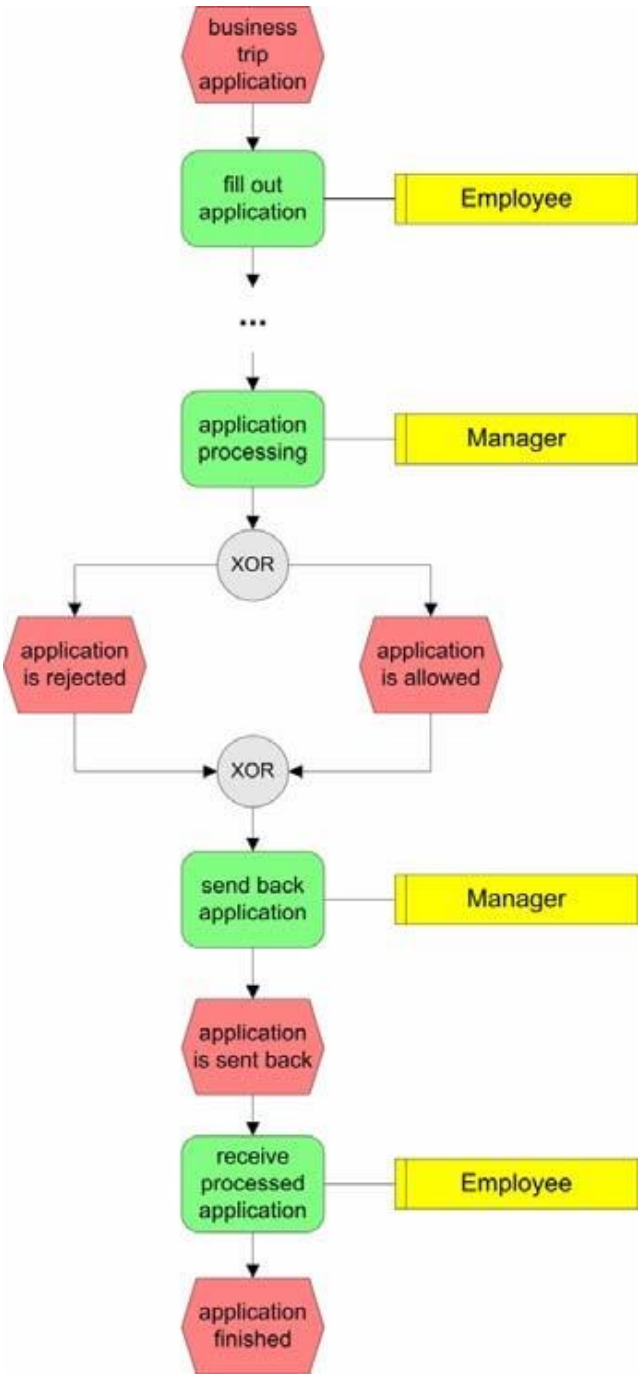


Figure 1.2. Example EPC for a Business Trip Application Process (taken from [97])

EPC is based on the concepts of stochastic networks and Petri nets, however using the EPC notation does not require a strong formal framework. The notation does not rigidly distinguish between output flows and control flows (as oppose to BPMN) or between places and transitions, as these often appear in a consolidated manner. Perhaps it was precisely this kind of simplification that led to the successful adoption of EPCs in practical applications [78]. Overall, EPC is a well-defined standard for domain modeling and it has been widely used for software engineering. It is not clear

how useful EPC would be for a domain-modeling task when the main goal is domain analysis with no intention to produce software. When comparing EPC to BPMN research [97] found that students preferred EPC when modeling their domain, but when they had to understand a domain modeled by someone else they preferred BPMN. The deeper analysis shows that BPMN, in spite of its subjective perception, leads to a more stringent modeling, which diminishes the number of the modeling errors [97]. The conclusion was that the reasons for this stricter modeling are the provided pools or lanes and the general absence of intermediate events. However, comparing to writing Use Cases both EPC and BPMN modeling is actually quite heavy and complex.

1.2.3. Topological Functioning Model and TFM4MDA

TFM offers a formal way to define a system by describing both the system's functional and topological features [53]. TFM is represented in the form of a topological space (X, Θ), where X is finite set of functional features of the business system under consideration (the business domain), and Θ is the topology that satisfies axioms of topological structures and is represented in the form of a directed graph [53]. Topology is a collection of open subsets that satisfies Kolmogorov's axioms of topological spaces [37]. TFM represents the system in its business environment and shows how the system is functioning, without details about how the system is constructed. TFM4MDA method developed in Riga Technical University [62], [64], [61] and [57] allows system's TFM to be composed by having knowledge in a form of informal description about the complex system that operates in the real world. TFM4MDA suggests using TFM as CIM within MDA; acquiring a mathematically formal and thus transformable CIM.

TFM has topological and functional properties. The topological properties are connectedness, closure, neighborhood and continuous mapping. The functional properties are cause-and-effect relations, cycle structure, inputs and outputs. These properties state the following [5], [53]:

- Separation of topological model of system functioning from topological space of an environment system works within. They also define formal semantics of concepts of subsystems and independent systems.
- In point of fact of functioning the common thing for all systems (technical, business and biological) should be at least one oriented cycle (a directed closed path). This property of the model enables analyzing similarities and differences of functioning systems.
- The model with any complexity can be abstracted to the simpler one and vice versa (continuous mapping in action). Because of this a lack of knowledge about the system can be filled up with knowledge that is obtained from the continuous mapping of the same type to the system model under consideration.

TFM4MDA approach suggests not avoiding the CIM, but instead starting the transformations from this model, thus providing evident consistency of PIMs. The TFM contains knowledge about its business functions, organizational units (positions rules, subsystem, units), and relations between business functions and between business functions and organizational units. TFM4MDA consists of the following steps: 1) retrieving the system's objects and functional features by analyzing the informal description of a system; 2) constructing a TFM's topological space using the retrieved system's objects and functional features; 3) constructing a TFM's topological graph using its topological space; 4) verifying the functional requirements by mapping them to the corresponding functional features; 5) transforming TFM to UML.

Apart from TFM being a mathematically formal model, there are some other benefits of using TFM for domain modeling. One of them is the cycle. The cycles provide a possibility to validate the TFM against the actual functioning system in real life. It is possible to trace through the functional features and see if the cycle makes sense. This provides possibilities to do formal validation of the domain model by doing main cycle and sub-cycle analysis. Another benefit of TFM is its inputs and outputs. Inputs and outputs are used to define the scope of the TFM and separate it from its environment. Separation is done formally with a closure operation. Separating the business system from its environment is handy to identify dependencies and interactions to other systems.

However, TFM and TFM4MDA have its downsides as well and these the author is going to analyze in the beginning of Section 2. Some of these are as follows. Constructing a TFM is not an easy task and TFM4MDA is a heavy approach with manual steps. TFM4MDA depends on an informal description of a business system, which might not exists in the first place. And if it is required for the system analyst to create this informal description, it is not clear if it would be worth it. Since he could use some other approach and save time. There is also no tool support for TFM4MDA as of now.

1.2.4. Ontology

To someone who wants to discuss topics in a domain D using a language L, Ontology provides a catalogue of the types of things assumed to exist in D; the types in the Ontology are represented in terms of the concepts, relations, and predicates of L [47]. Some reasons for developing an Ontology are: 1) To share common understanding of the structure of information among people or software agents; 2) To enable reuse of domain knowledge; 3) To make domain assumptions explicit; 4) To separate domain knowledge from operational knowledge; 5) To analyze domain knowledge [24]. Ontologies are used for different purposes, however authors research focuses on Ontologies developed for a business domain, describing business terms and their relationships. Ontology defines the terms used to describe and represent an area of knowledge. Ontologies include computer-usable definitions of basic concepts in the domain and the relationships among them.

Authors of the "YAGO: A Large Ontology from Wikipedia and WordNet" [94] say that state-of-the-art formalism in knowledge representation is currently is the Web Ontology Language (OWL). However, they decided to introduce a new model based on RDFS (Resource Description Framework Schema), which is the predecessor of OWL, for the reason of it no supporting transitivity. The example provided in the paper [94] is that it is not possible to define "Elvis Presley won Grammy Award in 1967" with OWL. However, the author argues that it is possible to do it with OWL by class hierarchy. For example, if there is a class "GrammyAward" there can be a sub-class "GrammyAward1967". Thus one can define an OWL property "hasWonPrize"

between classes “ElvisPresley” and “GrammyAward1967”. Furthermore, one could even define a property “wasGivenIn” for classes “GrammyAward1967” and “1967” (or individual if applicable).

OWL [90] is a common standard for defining Ontologies and will be considered for further knowledge integration for domain modeling. Ontology built according to the OWL standard can consist of classes, object properties, data properties, annotation properties, data types, and individuals [70]. Class is a collection of objects, which can contain any number of individuals. Classes are organized in a class hierarchy, so a class can be a sub-class or a super-class to another class. Data types are the same as classes, but can only contain data values, not individuals. Property is a directed binary relation that specifies a characteristic of a class. Object properties define relations between individuals. Data properties define relations between individuals and attributes. Annotation properties are used to provide annotations for Ontology. Individuals represent actual objects from the domain – instances of a class [6].

Ontologies are further discussed in section 2 as a tool for capturing declarative knowledge. From the perspective of domain modeling Ontology provides means to define the structure of the domain and OWL enables a formal way to do it. However, Ontology is not enough since it does not describe the domain from the perspective of processes – events and their sequence. You can build so called Process Ontologies [95], however this purely described only the components of a particular process and relationships between the components. In [95] the authors describe a Cutting Process Ontology, which consists of three sets of first-order axioms: Shape Ontology, Shape Cutting Ontology, Cutting Process Taxonomy. Although the Process Taxonomy does describe a set of rules how the process ends, these Ontologies do not describe the process with events and their sequence. So you do not know how the process starts, and what steps need to be executed to reach the goal. Structure is not enough; it is also necessary to know the processes or how to use the structure to do something. Nevertheless, the power of Ontology for defining the concepts and their relations within a domain should not be underestimated.

1.2.5. Use Cases

Use cases are useful in capturing and communicating functional requirements, and as such they play a primary role in product definition [39], or the business domain scope from a computation independent perspective. Ivar Jacobson in [29] clarifies a misconception of Use Cases that they are only applicable to user-intensive systems where there is a lot of interaction between the human users and the system or only for software development. In fact, just as useful for embedded systems with little or no human interaction as they are for user intensive ones, and also the application of Use Cases is not limited to software development. They can also help to understand the business requirements, analyze existing business processes, design new and better business processes, and exploit the power of IT to transform your business. The author wants to stress this misconception noted by Ivar Jacobson, in fact use cases can be used to describe business processes. A business process can be defined as a set of activities or tasks that need to be completed for an organizational goal [100]. A Use Case step can represent these activities or tasks.

Each Use Case defines a goal-oriented set of interactions between external actors and the system under consideration. A Use Case is initiated by a user with a particular goal in mind, and completes successfully when that goal is satisfied. It describes the sequence of interactions between actors and the system necessary to deliver the service that satisfies the goal. It also includes possible variants of this sequence, e.g., alternative sequences that may also satisfy the goal, as well as sequences that may lead to failure to complete the service because of exceptional behavior, error handling, etc. [39]. As stated in [10] one of the more surprising things about the UML standard is the lack of detail on the structure of Use Cases. A Use Case can be described in plain text, using operations, in activity diagrams, by a state-machine, or by other behavior description techniques, such as pre-and post conditions. The interaction between the Use Case and the actors can also be presented in collaboration diagrams. However no format is presented for any of these alternatives. Moreover, in this thesis Use Cases should not be confused with the UML Use Case Diagram, which is a simple representation of actors and Use Cases interaction, but does not include the detailed Use Case definition.

Usually Use Cases consist of the following elements – identifier, description (goal to be achieved by the Use Case), actors, assumptions (this could be split into pre-conditions and post-conditions), steps (the main scenario of events), variations (alternative scenarios), non-functional requirements, and issues (list of issues that remain to be resolved). Nevertheless, Use Cases have many different templates and there is no standard one way to describe them. Some of the steps of creating Use Cases for a domain includes the following: 1) identifying actors of the domain – users, user groups, system components (in a computation independent perspective these would be the business roles); 2) identifying particular goals for each role that the business domain supports; 3) creating Use Cases for each of the goals and maintain the same level of abstraction throughout a single Use Case (However, steps in higher-level Use Cases may be treated as goals for lower level Use Case); 4) reviewing and validating Use Cases with the stakeholders.

Although Use Cases are one of the most popular techniques for describing system functionality, Use Cases are also used to explore their non-functional characteristics [39]. For example, it is possible to relate performance requirements to the time taken between specific steps of a Use Case or to list the expected service levels for a Use Case as part of the Use Case itself. Use cases will be explored in more detail in Section 2 and used to describe the procedural knowledge of a domain. Overall Use Cases are a very easy to use approach for capturing the business process of a domain, which requires almost no training. Use cases are understood by the technical people as well as by the business people, so it is a great tool to be used at the beginning of software engineering or domain analysis. However, at later stages there might be too many Use Cases to trace the whole business process, so other means of domain modeling should probably be introduced.

1.2.6. Linguistic Assistant for Domain Analysis

Linguistic Assistant for Domain Analysis (LIDA) enables the system analysts to develop an object-oriented model for a business domain. Usually to acquire a domain model the system analysts or knowledge engineer need to analyze large volumes of text from legacy documents – these might include user manuals of legacy

systems, company policies, Use Cases, or transcripts of interviews with domain experts [68]. LIDA is able to process these texts to help the analyst identify the objects and model elements. By also providing a model-editing environment the model elements are refined through a validation process. Some of the features of the toolset include – domain-independent linguistic processing (identification of noun, verb and adjective), type assignment to words and multi-word terms (e.g. class or attribute), the evolving UML model is displayed in parallel automatically, and the model can be exported for use in other tools [69].

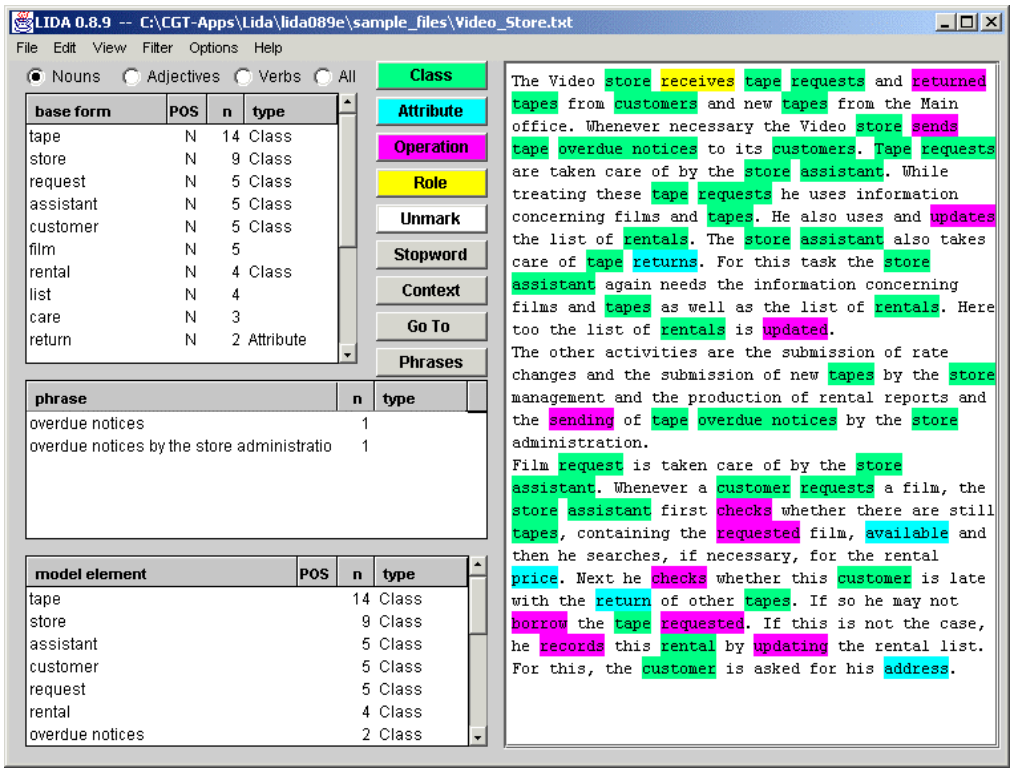


Figure 1.3. LIDA's Text Analyzing Environment

LIDA thus provides reliable tools for the user to analyze texts, but it does not analyze texts itself. The current state-of-the-art in natural language processing does not allow for dependable deep semantic analysis or even for adequate syntactic analysis; we leave it up to the user to find important information in the texts, to notice inconsistent or incomplete information in the texts, and to resolve these inconsistencies and omissions in the modeling phase [69].

This approach provides a very handy toolset for a system analyst, however the models still have to be manually constructed. The resulting model is a UML Class Diagram, which usually is considered as the part of the solution model (that can be used as part of a technical specification for software development). Author concludes

the LIDA approach to skip an abstraction level. In this approach the system analyst models the solution, but skips the domain analysis altogether. However, the natural language processing applied seems to be very handy for the system analysts and surely helps to deal with the domain knowledge at hand.

1.2.7. Natural Language Requirements Analysis

Approach discussed in [20] called NIBA (natural language requirements analysis in German) is also based on Natural Language Processing (NLP). Natural language requirements specifications form the basis for the subsequent phase of the information system development process, namely domain modeling. First the textual specifications are linguistically analyzed and translated into a so-called conceptual predesign schema KCPM (Klagenfurt Conceptual Predesign Model). This is formulated using an Interlingua, which is based on a lean semantic model, thus allowing users to participate more efficiently in the design and validation process. After validation, the predesign schema is mapped to a conceptual representation (e.g. UML). The sequence of these translation and transformation steps is described by the ‘‘NIBA workflow’’ [20]. KCPM offers a set of semantic concepts for modeling static and dynamic domain aspects. It is based on Ontological approaches, which treat the domain as a system consisting of interrelated elements (things) that are able to perform services (operations) and that are activated by events (messages sent by other things). Thus, there is a strong relationship to object-oriented approaches, which allows for a rather natural way to map KCPM concepts to OOA concepts [40].

The main modeling notions of KCPM are: thing-type, connection-type and perspective. Thing-type can be natural or juridical person, material or immaterial object, abstract notion, or a descriptive characteristic, e.g., “author”, “book”. Connection-type enables to define relationships between thing-types, e.g., “write”. Perspective defines the point of view of the involved thing-types, e.g., “Authors write books. (Perspective of author)” and “Books are written by authors. (Perspective of book)”. The data for KCPM is put into glossaries or information dictionaries. Conceptual predesign precedes the conceptual design and allows a more user-centered way for requirements elicitation, analysis and validation. To fill up KCPM glossaries

is a cumbersome and time-consuming task even if supported by appropriate tools, thus the research exploits NLP to extract glossary entries automatically from natural language requirements specifications [40].

Currently the approach only supports natural language specifications in German. This specification is then analyzed with NLP to acquire the KCPM glossaries, which can be considered as the domain model. One of the NIBA objectives is to automate as far as possible the mapping from the predesign scheme entries to a first cut conceptual scheme, e.g. formulated with the means of UML. Thus, a system analyst must not 'think' in UML-terms when analyzing requirements but can focus on collecting and relating glossary (i.e. predesign schema) entries representing relevant domain model aspects [40]. In authors opinion this is a very interesting objective, since there much more details necessary on the UML level for the solution model. Thus, if a system analyst focuses on the solution model he can loose sight of the domain model, which can result in a corrupt domain model. One of the problems author sees with the NIBA approach is the poor integration of the KCPM with MDA. It is good that the researchers goal is to acquire UML, which complements MDA. However, the KCPM itself and the supporting toolset is custom built and it is not clear what guidelines have been followed. Another problem with this approach is the obvious – where should the system analyst get the specification in natural language in the first place? If it doesn't exist the system analyst has to write it down, and then it is not clear if it is more efficient to actually write it down or choose another approach to capture the scope of domain.

1.3. Model Driven Architecture

OMG is considered to have one of the best standardization processes [86]. OMG members produce each new specification in nine to seventeen months. This process is proven in the marketplace with the standardization of CORBA, Domain Facilities, UML, the MOF, and OMG's other modeling standards.

MDA is an approach to using models in software development proposed by OMG. They consider MDA as a small step on the long road to turning software development from a craft into an engineering discipline [43]. MDA is neutral from

language, vendor and middleware [35]. It is based on the OMG standards. UML, the Unified Modeling Language, allows a model to be constructed, viewed, developed, and manipulated in a standard way at analysis and design time. As blueprints do for an office building, UML models allow an application design to be evaluated and critiqued before it is coded, when changes are easier and less expensive to make. In addition, the Common Warehouse Meta-model (CWM) standardizes how to represent database models (schema), schema transformation models, OLAP and data mining models, etc. The Meta-Object Facility (MOF) standardizes a facility for managing models in a repository. And finally, XML Metadata Interchange (XMI) is an interchange format for models, based on the MOF and the popular language XML. At a conceptual level MDA is a holistic approach for improving the entire IT life cycle – specification, architecture, design, development, deployment, maintenance, and integration – based on formal modeling [58]. MDA is a framework of technical standards progressively being developed by the OMG. J. Osis believes that the potential power of MDA is in model transformation, because model transformation requires a use of formal languages for description of models and this lead to increasing the role of mathematics for software development [54].

OMG describes different levels of abstraction and their relations, however it does not specify how to create the models and which notation to use for representation. There are some recommendations from various researchers in the field that can be similar in certain points and different in others [33]. MDA defines 3 viewpoints and the corresponding models – Computation Independent Model (CIM), Platform Independent Model (PIM), Platform Specific Model (PSM).

1.3.1. Computation Independent Model

CIM represents the viewpoint that focuses on environment of the system, and the requirements for the system. This level of abstraction represents business processes and structure of the organization for which a software solution might be developed. The business analysts, domain experts or domain users of the system, should understand the CIM. It describes the environment that an information system should operate in and it helps to recognize the required functionality. The CIM plays an

important role in bridging the gap between those that are experts on the domain and its requirements on the one hand, and those that are experts of the design and construction of the system artifacts that together satisfy the domain requirements, on the other [43]. Since the domain model is the main focus of this work, the author has discussed some domain modeling approaches that are available in section 1.2.

1.3.2. Platform Independent Model

Ideally, developers should see a computer system as an independent world during its development, without any artificial constraints such as the operating system, hardware, network performance, or application incompatibility [25]. PIM represents the viewpoint that focuses on operation of a system while hiding the details necessary for a particular platform. Thus PIM does not change from one platform to another. A very common technique for achieving platform independence is to target a system model for a technology-neutral virtual machine [43]. Here OMG points out that by using such virtual machine it is possible to be independent from platform as in operating system. However, such a perspective would not be independent from the technology platform. For example, when using a Java Virtual Machine it is independent from the operating system, but it is still tied to the Java technology. Based on the same PIM it should be possible to implement a C# based solution, if decided so by the stakeholders (this could happen do to budget or strategy reasons for example). PIM should also be independent from the technology used. PIM describes required services of the computer system, but hides details in usage of concrete technology. PIM is defined by the use of UML.

1.3.3. Platform Specific Model

PSM represents combination of the platform independent viewpoint with an additional focus on the details for a specific platform. PSM gives specific details for the computer system and the type of platform used. Developers need to take part of creating this model. Within MDA life cycle the PSM model can be transformed into code, since it has all the necessary information. PSM is defined by the use of a specific

UML Profile, for example, UML Profile for Enterprise Java Beans (EJB), C# or any other platform.

1.3.4. Iterative Development

A requirements model (CIM) is more abstract than PIM, which is more abstract than PSM. People often assume that you complete the more abstract models before moving on to define more concrete ones. However, that is not actually the case, because a scalable model driven development process requires revising artifacts at all levels of abstraction throughout the process [23]. Model-Driven Development (MDD) is an iterative process as oppose to a waterfall process, which consists of strictly sequential steps. Each level of abstraction in MDA provides a different viewpoint. And in some viewpoint a modeling inconsistency with the problem domain can be more obvious that in another, so it is possible to return to that viewpoint and do the correction. Moreover, the different abstraction levels will be built at different points in time, so additional information can come in a later phase and it is always possible to return to an earlier one. This ensures that the models are as up-to-date as possible since inconsistencies with the problem domain will cause issues for the analysis or planned solution. Changes can come at any time to any abstraction level, so it is very important for the models to be bound by model transformations between each other. Thus changes in one abstraction layer can be depicted in the others, if these are relevant for the particular level. This iterative development process is one of basic principles of MDD and reflects on all artifacts in MDA life cycle. For example, if CIM would consist of artifact A and B and there would be a transformation from A to B, then it is accepted to do changes in A while at the same time modeling B.

1.3.5. The Meta-Object Facility

A metamodel uses Meta-Objects Facility (MOF) to formally define the abstract syntax of a set of modeling constructs. MOF can be used to define and integrate a family of metamodels using UML class modeling notation to describe MOF compliant metamodels [50]. MOF proceeds from the basic premise that there will be more than

one kind of model and therefore that there must be more than one modeling language. The MOF architects saw that the industry was using completely different means for describing the nature of different kinds of modeling constructs [23]. In section 1.2 the author looked at some of the domain modeling approaches and each of them have their own modeling constructs. However, only BPMN, TFM and OWL have a metamodel formally defined according to MOF. Other approaches lack this level of formalism and this is a big issue in terms of integration with other models and transparency. A true modeling tool enforces the semantics and syntax. Moreover, pictures created, for example, with Microsoft PowerPoint or Microsoft Visio have no formal semantics or syntax [3].

MOF is a universal way of describing modeling constructs and its architecture consists of 4 "metalevels" – M3, M2, M1, and M0. Level M3 is the MOF itself or the meta-metamodel. It provides the means to model other metamodels. Level M2 is the metamodel, which is defined via the MOF constructs. For example, this can be a metamodel of UML, BPMN or TFM. M2 constructs are instances of M3 constructs. Level M1 is the model. At this level the model consists of instances of M2 constructs. For example, this can be a particular UML model for a library system, which has an element "Librarian" (class). Level M0 consists of objects and data, which are instances of M1 elements. For example, consider a person "Jānis Bērziņš" who is a librarian. At level M0 this is an instance of the M1 element "Librarian".

A question might arise – is MOF a "cure for all diseases"? At the moment there are no real alternatives, but anyone who is reluctant to use MOF for their model or modeling language is free to propose their own meta-meta-model. Of course, a formal transformation between MOF and this potential new meta-meta-model would still be required; otherwise there would be no use of it (and conformant modeling languages would not be able to integrate still). Anyway, in author's opinion this discussion is a waste of time and would only make sense if some serious problems with MOF would be discovered, but until then any model or modeling language should use MOF.

1.3.6. XML Metadata Interchange

The XML Metadata Interchange (XMI) is widely used XML interchange format, which defines the following aspects involved in describing objects in XML [92]: 1) The representation of objects in terms of XML elements and attributes; 2) The standard mechanisms to link objects within the same file or across files; 3) The validation of XMI documents using XML Schemas; 4) Object identity, which allows objects to be referenced from other objects in terms of IDs and UUIDs (universally unique identifiers); 5) XMI describes solutions to the above issues by specifying rules to create XML documents and Schemas that share objects consistently. XMI defines how to represent models as XML documents in a common way.

UML is the most well known MOF metamodel, thus XMI Document Type Definition (DTD) for UML is the most well known XMI DTD [23]. Most UML tools support importing and exporting UML models as XMI, which are validated against this DTD. However, one should treat with caution when a vendor claims that their tool supports XMI. Since it might only mean that the XMI DTD for UML is supported. More comprehensive compliance with MOF/XMI would entail being able to act as a generator that produces an XML DTD or Schema from any arbitrary metamodel's abstract syntax [23]. One example of such implementation is Eclipse EMF [17]. The author is going to use it to develop a toolset in Section 3.

1.3.7. Model to Model Transformation

There is a range of tool support for model transformation using different mixtures of manual and automatic transformation. The manual transformation process is not a lot different from how good software design work has been done for years [43]. However, the MDA approach adds value in two ways: 1) the explicit distinction between a platform independent model and the transformed platform specific model; 2) the record of the transformation. Nevertheless, since MDA introduces formal means to define the models (e.g. MOF/XMI) it is also possible to enable some degree of automation for model transformation.

In MDA the model transformation is achieved by using mappings. A mapping is a set of rules and techniques used to modify one model in order to get another model. Mappings are used for transforming: PIM to PIM, PIM to PSM, PSM to PSM and PSM to PIM [44]. PIM to PIM is used when models are enhanced, filtered or specialized during the development life cycle without needing any platform dependent information. PIM to PSM is used when the PIM is refined to be projected to the execution infrastructure. Platform specific characteristics should be defined with a UML profile. PSM to PSM is used for component realization and deployment, specifying initialization data, target machines, configuration, etc. PSM to PIM is used to acquire a platform independent viewpoint of existing implementations with a particular technology. Moreover, if a formal CIM exists then also similar transformations with CIM are possible. According to [5] there are at least 2 transformations within CIM itself: transformation from CIM-Knowledge Model to CIM-Business Model, and transformation from CIM-Business Model to CIM-Business Requirements for the System. This structure of the CIM is defined by the authors of [5] and can differ on the viewpoint; however, it is interesting to note that such a structure exists within CIM. Models used within MDA transformations should be formal, meaning, to be understandable by a computer and transformable. The OMG proposes model transformations starting from the PIM [44], since the CIM is considered rather informal then formal. The author will demonstrate how a formal CIM can be constructed and model transformation developed within CIM in section 2.

Query/View/Transformation (QVT) is an OMG standard for model transformations, which consists of several model transformation languages. The QVT specification has a hybrid declarative/imperative nature, with the declarative part being split into a two-level architecture [42]. Declarative part of the QVT consists of the Relations and Core languages, which embody the same semantics at two different levels of abstraction. Relations metamodel and language supports complex object pattern matching and object template creation. Core metamodel and language is defined using minimal extensions to Essential MOF (EMOF) and Object Constraint Language (OCL). Imperative part of QVT consists of the Operational Mappings language and the Black Box. Operational Mappings (QVTo) is specified as a standard way of providing imperative implementations, which populate the same trace models

as the Relations language. Finally, the Black Box provides a possibility to implement custom mapping and plug it into the model transformation. The author is going to use QVTo and Black Box for a model transformation in section 3.

1.4. Domain Model within Model Driven Architecture

In Section 1.1 the author has given a detailed definition of the domain model and here the author looks at the domain model from perspective of MDA. Domain model and CIM is not the same thing, even though in context of MDA it is often considered to be one and the same. MDA's CIM describes the same viewpoint as the domain model, however CIM usually has a smaller scope and is intended for depicting requirements for a software solution. A domain model on the other hand can exist without such requirements and serve for defining a business and doing analysis. Another issue discussed is that there is no standard way to define a CIM and do a transformation to PIM.

1.4.1. Computation Independent Model versus Domain Model

David S. Frankel in [23] describes distinctions between domain versus system model and requirements versus computation model. A domain model describes aspects of the business, irrespective of whether those aspects are going to be automated. On the other hand, a system model describes aspects of computer system that automate elements of the business. The scope of a domain model usually is bigger than the scope of a system model. This is shown in Figure 1.4. The big circle is everything that there is as an abstract domain scope. The cloud inside represents a defined domain model, which is a specific part of everything with a defined scope. For example, this could be a domain model for a particular Library. The rectangle inside the cloud is the CIM, which is a particular part of the whole domain model that is a scope for a planned or existing software solution. Moreover, the lines of the cloud and the rectangle represent the scope borders. There will always be a predefined scope and interaction from within the scope to the outside via an interface. If there is a software solution based on the CIM, then there are also related processes in the domain model

outside the CIM that this solution interacts with. For example, in a Library information system the Librarian gives out a book to a client. From perspective of the Library information system the process has ended and there will be another process started when the book is returned. But at the domain model level the process continues, because the client is going home and reading the book, and then bringing it back. Thus, we can also identify inputs and outputs of the particular CIM or domain model (on a higher level to interaction with "everything").

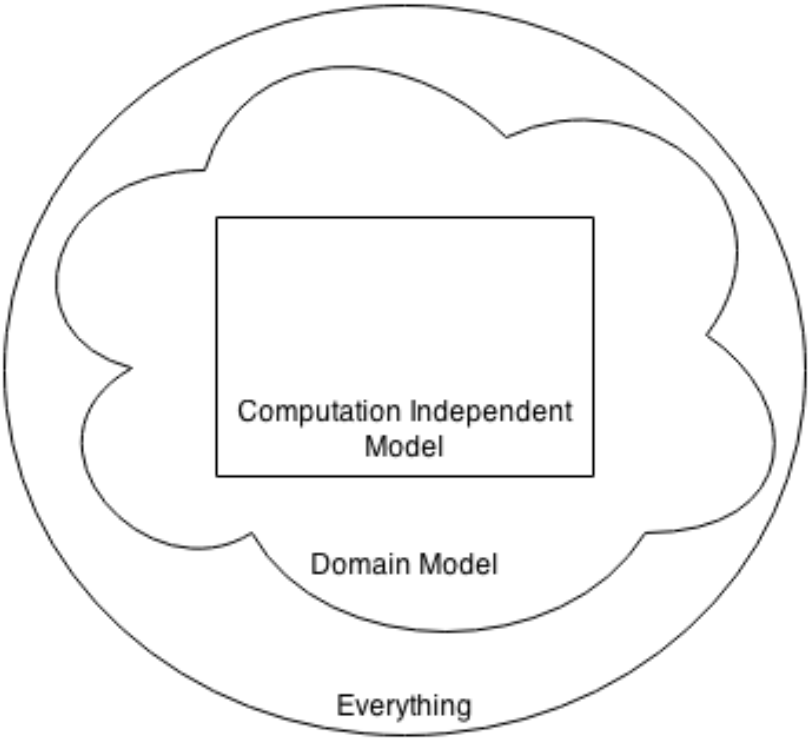


Figure 1.4. Domain Model vs. CIM

A requirements model (CIM) describes the logical system rather than the business, but does so in a computation-independent way – technical factors are not taken into account. Computational models, like PIM and PSM, describe the logical system as well, but take technical factors into account. OMG defines CIM as the requirements model [43]. Requirements engineering analyzes the problem domain (domain model) and describes the solution domain (requirements model or CIM according to David S. Frankel). There is a large set of requirements gathering methods available and usually functionality for a TO-BE system is defined based on business goals. This can lead for the functionality to be fragmentary defined from the problem domain and the problem domain information is mixed with the solution information [58]. The author agrees, that this is a one of risks of doing domain modeling based on

requirements, because requirements come only after there is a need for a software solution. If we would want to acquire an independent problem domain, this would need to happen before there are any requirements for specific information systems. And ideally a domain model would already be in place so that a system analyst would only need to mark the CIM within this domain model.

1.4.2. How to Model the Computation Independent Model?

CIM is a precise description of the business in its environments, in a language that business people can understand and is dedicated for business purposes [5]. However, there are different opinions of what should be modeled in CIM. Research [35] summarizes some of the meanings of CIM that are used: 1) environment of the system and requirements for the system; 2) a business model; 3) a domain model; 4) business requirements; 5) knowledge about the problem domain; 6) semantics of business vocabulary and business rules; 7) BPM. There are different opinions on what the CIM is and how to define it. CIM is the most abstract, but at the same time most domain specific model level within MDA, which may be modeled using any kind of domain specific description [10]. The obscure method to define CIM might be regarded as advantage, since it offers much flexibility, or as disadvantage, since it avoids efficient tool support to specify and transform the CIM to the next model level. Thus, a CIM is extended towards a PIM by hand by enriching it with operational model elements, which in general involves multiple human interpretations of the imprecisely described CIM. Then, using the MDA often results in a semantic gap between the CIM and the PIM [10]. In authors opinion the way to solve this is to provide a formal means to define a CIM, as well as a formal transformation to PIM.

1.5. Evaluation of Domain Modeling Approaches

There are, of course, a lot of domain-modeling approaches and the author has only looked at a few (in section 1.2). However, these approaches give an indication of how the domain model can be modeled today. Domain modeling approaches are going to be evaluated based on 3 sets of criteria: 1) criteria for a formal domain model; 2)

criteria for conformance to MDA; and 3) criteria for practical usability. Weights of all criteria are subjectively set to be equal, based on authors experience with domain modeling and software development.

1.5.1. Criteria for a Formal Domain Model

Formalism is a theory or a view of philosophy of mathematics as per Nicolas D. Goodman [46], which states that mathematics is a rule-governed manipulation of symbols and nothing else. Thus this manipulation is formal. How to evaluate weather a domain model is formal? A metamodel does provide a level of formalism since it defines the rules for defining the elements of the domain model. If the metamodel complies with a meta-metamodel, this brings even more formalism. To analyze the formalism of the domain modeling approaches the author has defined 3 criteria in table 1.1. The first criterion is a mathematically formal model. Mathematically formal model is a step further than a metamodel, since the rules how to build the model are defined by mathematics and/or mathematical logic. The second criterion for a formal domain model is a formal scope definition. Domain modeling approaches provide syntax and semantics to build models, thus defining what elements are available and how to use them, and also the meaning of the elements. However, another important dimension is the scope of a domain model, which defines the borders of the domain model. The scope is usually left up to the system analyst to decide and thus is defined intuitively. Formal scope definition means that formal rules exist to determine the scope. The third criterion for a formal domain model is formal model validation. After the domain model is built there have to be procedures how to ensure it's validity before doing further transformations. The value range for all 3 criteria is "Yes" or "No". The author evaluates the approaches in table 1.1. and provides explanation below.

Although BPMN, EPC and NIBA do have a metamodel in general, the model itself is not mathematically formal model because the rules to build this model are not defined by mathematics. On the other hand, TFM is based on mathematics and thus is considered to be a mathematically formal model.

1.1. Table

Formality of Domain Model for Domain Modeling Approaches

Criteria	**Value Range**	**BPMN**	**EPC**	**TFM**	**Ontology**	**Use Cases**	**LIDA**	**NIBA**
Mathematically formal model	Yes/No	No	No	Yes	Yes	No	No	No
Formal scope definition	Yes/No	No	No	Yes	No	No	No	No
Formal model validation	Yes/No	No	No	Yes	Yes	No	No	No
Score	Score from 0 to 3	**0**	**0**	**3**	**2**	**0**	**0**	**0**

Ontology is based on mathematical logic or predicate logic, thus the author considers this also a mathematically formal model. Another examples of a mathematically formal model would be Petri Nets [27]. The scope of a TFM is formally defined by the closure procedure based on input and output analysis. None of the other approaches provide a formal way to do this. Thus only TFM has formal scope definition. Validation of the model for a TFM is done by cycle analysis and the main rule is that there should be at least one cycle for the system to be functioning. Since Ontology is based on predicate logic it is possible to validate it with querying the Ontology. The knowledge engineer can ask questions to the Ontology can check weather he gets the answers he expects. This is done with a semantic reasoner. Thus, Ontology is considered to also have a formal model validation. Again none of the other approaches provide similar possibilities other than transforming the domain model. After model transformation based on the outcome it is also possible to evaluate the validity of the model, however this is not considered as formal model validation by the author in context of this evaluation. Model transformation is one of the criteria of conformance with MDA evaluation.

1.5.2. Criteria for Conformance with Model Driven Architecture

The goal of this section is to show an example on how conformance of a domain modeling approach to MDA can be measured. Also this will show the conformance of some of domain modeling approaches and discuss the consequences. Thus, the author wants to stress the importance of conformity to MDA. Of course, some of the domain modeling approaches were never intended to conform to MDA or were developed before there was MDA.

To measure conformance of domain modeling approaches the author proposes the following 3 criteria: computation independence, MOF compatibility, and model transformation to PIM/PSM. Since the discussion is about domain modeling, thus the corresponding MDA's level of abstraction is CIM. The domain model has to correspond to the computation independence viewpoint of MDA. Compatibility to MOF is key to state that a model is formal and transformable within MDA. And last but not least, within MDA there has to be a formal way to transform the CIM to PIM/PSM. Thus, mapping/transformation from the domain model to UML has to exist. Author's rule for the transformation criteria is that if there is a transformation from model A to UML directly then it is a "Yes", if there is an indirect way of getting to UML like via model B then it is a "No". Same rule applies for MOF compatibility (model has to be directly be compatible with MOF). Each domain modeling approach was assessed according to these criteria with "Yes" or "No". "Yes" – means the approach is compatible with this aspect of MDA, and "No" is the opposite. There is also a score, which counts how many criteria is "Yes". The assessment can be seen in table 1.2.

As it is possible to see in table 1.2, BPMN is fully conformant with MDA with all 3 criteria true. BPMN is one of OMG standards, however other researchers do the transformation to PIM/PSM. Authors of [52] describe a transformation from BPM to UML Activity Diagram. Business Process Execution Language for Web Services (WS-BPEL) is a standard for implementing business processes on top of web services [26]. There is a model transformation from BPMN to BPEL, thus acquiring working software based on web services via modeling [67]. However, in author's opinion this

is jumping from CIM to PSM level and skipping PIM. PIM level is still valuable for domain analysis and there has to be a possibility to acquire UML.

1.2. Table

Domain Modeling Approach Conformance to MDA

Criteria	**Value Range**	**BPMN**	**EPC**	**TFM**	**Ontology**	**Use Cases**	**LIDA**	**NIBA**
Computation Independence	Yes/No	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Compatibility to MOF	Yes/No	Yes	No	Yes	Yes	No	No	Yes
Model transformation to PIM/PSM	Yes/No	Yes	Yes	Yes	No	No	Yes	Yes
Score	Score from 0 to 3	**3**	**2**	**3**	**2**	**1**	**2**	**3**

On the other hand, Event-Driven Process Chains (EPC) is conformant from perspective of viewpoint; it does show the same abstraction level as the computation independent model. EPC's metamodel is not defined using MOF, thus to integrate with other models it would be necessary to do a transformation to an intermediate, MOF-compatible model (which is a drawback from a model usability and integration perspective). Earliest mapping attempts from EPC to UML include Use Case Diagram, Activity Diagram, and Class Diagram [98]. However, also more recent and well-defined transformation from EPC to UML Activity Diagram exists [99]. TFM has the computation independent viewpoint and it is possible to define a domain model and a CIM. TFM's metamodel is defined in accordance to MOF and it has a formal transformation to UML. Thus, the TFM is also fully conformant to MDA. Ontology does have the right viewpoint and in this context a domain model and a CIM can be modeled. OWL has a MOF compatible metamodel, but there is no standard transformation to UML. There have been attempts to transform UML to Ontology like in [25], but not the other way around. Use Cases can have the right viewpoint of computation independence, but there is no metamodel for the textual Use Cases (not to

be confused with UML Use Case Diagram). And there is also no standard transformation to UML. LIDA and NIBA both start with an informal description and natural language analysis is applied. The description complies with the computation independent model viewpoint. For LIDA there is no metamodel at this level at all, since after text analysis UML is acquired. NIBA does have a metamodel, and it also supports export to XMI in correspondence to MOF. NIBA does have a transformation to UML as well. Thus NIBA is fully compatible to MDA.

1.5.3. Criteria for Practical Usability

In addition to formal domain model and conformity to MDA there are also practical considerations for a domain modeling approach. To analyze the practical usability of the domain modeling approaches the author has defined 4 criteria in table 1.3. The first criterion is support for declarative knowledge. This defines weather the domain model describes the concepts and their relationships in the problem domain. For this criterion also the level of detail is considered and authors main rule is if it is possible to define the concepts in a single hierarchy only then it is a "Yes" (otherwise concepts and relationships are fragmented and do not provide enough details on the structure). The second criterion is support for procedural knowledge. This defines weather the domain model describes the functioning or procedures of the problem domain. From practical perspective it is of course important that both aspects of the problem domain are supported. The third criterion is tool support, which defines weather there is tool support for the domain modeling approach. The fourth criterion is popularity or familiarity of the approach. This is an important practical aspect for a domain modeling approach, because domain model has to be understood by the business people. This means that it is common in the IT area, thus it is easier to get familiar with it. Popularity will be measured based on the count of publications mentioning the approach in Google Scholar database [54]. This is not a direct indicator of popularity among practitioners, but a general popularity in the area of IT. Popularity does not mean that the approach is simple enough for the business people to understand. Complexity is another aspects that is very hard to measure, thus it is not

included in the evaluation. The value range for all 4 criteria is “Yes” or “No”. The author evaluates the approaches in table 1.3. and provides explanation below.

1.3. Table

Domain Modeling Approach Practical Usability

Criteria	**Value Range**	**BPMN**	**EPC**	**TFM**	**Ontology**	**Use Cases**	**LIDA**	**NIBA**
Supports declarative knowledge	Yes/No	No	Yes	No	Yes	No	No	No
Supports procedural knowledge	Yes/No	Yes	Yes	Yes	No	Yes	Yes	Yes
Tool support	Yes/No	Yes	Yes	No	Yes	Yes	Yes	Yes
Popular/Familiar	Yes/No	Yes	Yes	No	Yes	Yes	No	No
Score	Score from 0 to 4	**3**	**4**	**1**	**3**	**3**	**2**	**2**

From the analyzed approaches Ontology and EPC support declarative knowledge and, of course, only Ontology does this to full extent. However, it is hard to define procedural knowledge with Ontology, if possible at all. BPMN is not considered to support declarative knowledge because the concepts and relationships are fragmented and not represented in a single hierarchy. Pools and lanes of BPMN are not enough. For example, it is possible to define that Accounting Department has an Accountant and a Manager, but it is not possible to define the relationship of Accounting Department to the Sales Department with BPMN. It is possible to do this with EPC and thus it has an advantage to define the structure of the organization. Mostly domain modeling approaches are strong in procedural knowledge support. Tool support exists for all analyzed approaches except TFM. This is a major drawback of the TFM approach. For popularity a threshold of 1000 publications was introduced by the author to determine weather an approach is popular or not (thus “Yes” or “No” in the table 1.3). The publication count is as follows ordered in descending order: 1) Ontology – 1,250,000; 2) Use Cases – 98,500; 3) BPMN – 7,690; 4) EPC – 2,970; 5)

NIBA – 101; 6) LIDA – 60; 7) TFM – 57. This count s based on the Google Scholar [54] and the actual count maybe bigger, however it serves as a good enough indicator for a “Yes” or “No” value.

1.5.4. Final Evaluation Results

Based on the previous analysis here the author combines the 3 sets of criteria and presents the results of evaluation in table 1.4. This table analyzes the domain modeling approaches based on 3 criteria – formal domain model, conformance to MDA and practical usability. The value range of each criterion is based on the score from previous analysis in tables 1.1, 1.2 and 1.3.

1.4. Table

Final Evaluation of Domain Modeling Approaches

Criteria	**Value Range**	**BPMN**	**EPC**	**TFM**	**Ontology**	**Use Cases**	**LIDA**	**NIBA**
Formal Domain Model	Score from 0 to 3	0	0	3	2	0	0	0
Conformance to MDA	Score from 0 to 3	3	2	3	2	1	2	3
Practical Usability	Score from 0 to 4	3	4	1	3	3	2	2
Total Score	Score from 0 to 10	**6**	**6**	**7**	**5**	**4**	**4**	**5**

The strongest approaches from formalism perspective are TFM and Ontology. For conformance to MDA three approaches got highest score – BPMN, TFM and NIBA. From perspective of practical usability EPC scored the highest with BPMN, Ontology and Use Cases following. However, none of the analyzed approaches have reached the maximum possible score of 10. The closest to score of 10 is the TFM. TFM approach lacks in practical usability mainly because it does not have tool

support, does not support declarative knowledge and also is unfamiliar. The strong formalism comparing to other approaches is the main advantage of the TFM approach. Practical usability aspects can and should be improved in context of TFM research and this is one of the tasks of this thesis.

1.6. Summary

To summarize the author discusses drawbacks of the existing domain modeling approaches described above. Some of the problems include the following. BPMN is a widely used approach and has good integration with MDA and other approaches. However, BPMN is quite complex to start with and requires training. Also it is good for describing business processes, but lacks in describing business concepts and rules on its own. EPC is a bit better at defining structure since it enables the system analyst to create organizational chart (by defining organizations and positions in a single hierarchy). But again this approach is quite complex and requires training. Also it does not beat Ontology in defining concepts and their relationships, which would be the preferred approach for this task. On the other hand, Ontology does not propose anything to define events and their sequence (thus there is no way to setup a business process with Ontology alone). Use Cases is a simple approach – easy to learn and use. It can be well understood by technical people and also business people, which is an important advantage for domain modeling. However, Use Cases are expressed in natural language and thus can be inconsistent, ambiguous, hard to manage, and hard to transform. Also Use Cases lack the possibility to define a vocabulary (as oppose to Ontology). Approaches like LIDA and NIBA try to make use of NLP to enable a more efficient and transparent domain modeling based on textual artifacts in the beginning. As mentioned before these texts might not be available and it is not clear if production of these texts will be worthwhile. However, the use of NLP and the ability to trace the domain model back to the source of the information might be very useful. TFM is an uncommon approach and has a complex process of construction. There are no tools to support TFM4MDA and it also relies on the informal description of the business domain (text), which might not be there. Of course, it is always possible for the system analyst to produce the informal description, but would it make more sense to put the

gathered and analyzed knowledge in a more structured way if the system analyst has a choice? The author doubts the effectiveness of producing texts about a domain from scratch just to satisfy an entrance state of a domain modeling approach. Also there is currently no tool support for constructing a TFM and helping with TFM4MDA.

Based on the review of approaches only TFM and Ontology provide means to acquire a formal domain model. Thus, it would be possible to validate the scope and content in an automated way (with closure procedure and cycle analysis of TFM and predicate logic of Ontology). At the beginning of the domain modeling there should be a simple, but somewhat formal way to capture the declarative (structure, concepts, relationships) and procedural (business process) knowledge. After producing this knowledge in a formal way there should be means of transforming it into a more complex domain model. This domain model would be used as CIM in the context of MDA and further transformed to PIM/PSM.

2. THE INTEGRATED DOMAIN MODELING APPROACH

Based on the conclusions from Section 1 a new approach needs to be developed to acquire a formal domain model. This section introduces the Integrated Domain Modeling (IDM) approach developed by the author. The goal of this approach is to provide an efficient way to acquire a domain model based on declarative and procedural domain knowledge. This approach suggests using common system analysis and artificial intelligence practices to capture the domain knowledge and then transform these into a corresponding domain model. As per Section 1 Use Cases and Ontology are used for the domain knowledge and Topological Functioning Model (TFM) is used as the domain model. Use Cases are not mathematically formal, however this is a simple way of describing processes with minimal training and with the right Use Case template it can be possible to transform this to a more sophisticated model. Author does not propose a novel methodology for Use Case or Ontology development, but instead urges to reuse existing methodologies. The IDM approach provides a formal way to facilitate Ontology for software engineering by providing means to acquire a Computation Independent Model (CIM) within Model Driven Architecture (MDA). Since TFM is used as CIM the resulting domain model is mathematically formal and thus transformable. First author is going to discuss the solid basis of the IDM approach, which is the TFM, TFM4MDA and TopUML. TFM4MDA approach assumes that knowledge about a business system can be represented as an informal description. Since such an informal description can be far too complex, ambiguous, redundant and inconsistent for a formal analysis, the IDM approach is using formally defined knowledge with correspondence to well known standards – Ontology and Use Cases. The IDM approach improves the TFM4MDA approach for the purpose of acquiring a TFM.

2.1. Solid Basis for the Approach

This sub-section provides the solid basis for the IDM approach. The approach is based on the foundation of the Topological Functioning Model (TFM) and TFM4MDA approach, and complements the TopUML approach. The IDM approach deals and resolves the ambiguity of the construction of the TFM, which is vaguely addressed by the TFM4MDA and TopUML. Although author has discussed the TFM approach already in section 1.2.3 as part of the domain modeling, here the author is going to show the strengths and weaknesses of the TFM approaches and analyze potential improvements.

2.1.1. Topological Functioning Model Approaches

This work continues research on computation-independent modeling with TFM (discussed in section 1.2.3.) and specifically on TFM4MDA started in [79], [83] and [63]. As stated in [63] an informal description of the system in textual form can be produced as a result of system analysis. This informal description is a pre-condition for constructing TFM. The TFM4MDA approach proposes to transform a system's informal description into a TFM of the system. Of course, since it is an informal description – unstructured text in natural language – this transformation is completely manual and the responsibility of the system analyst. The approach described in [63] still defines some structure of the informal description, thus making it semi-formal. After an informal description is in place, the system analyst identifies system's objects and composes functional features. Every functional feature consists of an object action, a result of this action, an object involved in this action, a set of preconditions of this action, an entity responsible for this action, and subordination. Next step of the approach is to construct a topological space of TFM, meaning that the analyst has to identify the cause-effect relations between the composed functional features, define the main functional cycle and verify functional requirements.

Another branch of this research is suggesting a new UML profile – TopUML, which incorporates the topological nature of TFM with UML. UML approaches usually are strong with defining the class hierarchy, but TopUML provides unique

benefits for MDA, defining the Topological Class Diagram. By using TopUML it is possible to acquire cause-effect relationships between methods for PIM/PSM from CIM. Class diagrams reflect the statistic structure of the system, and by using class diagrams it is possible to model objects and their methods that are involved in the system. Using the standard UML class diagram specification it is not possible to reflect the cause-and-effect relations within a system to indicate which certain activity of an object triggers another objects certain activity. Uniqueness and main value of the Topological Class Diagram is that it is able to show the relations between different methods from the same or different classes [66]. This property can be used for PSM and code generation. It would potentially allow defining part of the logic within the methods of generated code, not only the structure. TFM for MDA approach allows doing this automatically by means of transformation from CIM. No other approach provides this possibility from perspective of CIM; this kind of logic has to be added manually by the PSM designer.

It is possible to develop a topological class diagram describing the business system, if the corresponding TFM has been developed. The transformation between TFM and topological class diagram, i.e., transformation between CIM and PIM/PSM is described in [12]. To execute the transformation several steps have to take place: 1) creation of the TFM; 2) creation of the problem domain objects graph; 3) transformation into class diagram. To obtain a problem domain object graph, it is necessary to detail each functional feature of the TFM to a level where it uses only one type of objects. After construction of problem domain object graph all the vertices with same type of objects and operations must be merged, while keeping all relations with other graph vertices. As a result, object graph with direct links is defined. This then can be transformed into a corresponding TFM. It is possible to develop a topological class diagram where the established TFM topology between classes is retained. In traditional model-driven development relations (mostly associations and generalizations) between classes in the UML class diagram are defined by the modelers' discretion.

2.1.2. Strengths of Topological Functioning Model

According to [5] the only formal means for definition of a domain model are Petri nets and their extensions. However, the researchers continue that also TFM can be considered as a formal approach for defining a domain model. The mathematical basis of the TFM is one of its main strengths. This is to be considered "light math" concerning relationship between objects. The functional features are based on connectedness, closure, neighborhood, and continuous mapping. TFM is a graph, but not any graph is a TFM [53]. By describing the functionality of a business system in its environments with functional features and cause-effect relationships, TFM provides a formal way to capture the procedural aspects of the domain model.

The second strength of the TFM is inputs and outputs, which correspond to system theory [65]. Inputs and outputs play an important role for domain modeling, specifically for identifying the scope of the domain model. The distinction between information system and its environments happens in the level of a domain model. Furthermore, in case of software engineering, it is then necessary to define distinction between the domain model and the requirements (part of the domain model would be the requirements or computation independent model and the rest its environment). For both situations inputs and outputs can be used to define the borders between the information system and its environments. It is also possible to analyze the dependencies of the system based on this. Moreover, input and output analysis gives the opportunity to validate the domain model from perspective of dependencies and scope.

Cycle structure within a TFM and the main functioning cycle is the third strength of TFM. From the perspective of functioning the common thing for all systems (technical, business and biological) should be an oriented cycle (a directed closed path) [60]. Every TFM needs to have at least one directed closed loop, but usually it is even an expanded hierarchy of cycles. This TFM's property enables analysis of similarities and differences among functioning systems [53]. Moreover, cycles again provide the opportunity to validate the domain model. By identifying the cycles is possible make sure that all functional features for the particular business process represented by the particular cycle are present. Also it is possible to analyze

the hierarchy of business processes – when a cycle consists of sub-cycles this creates a parent/child relationship (parent-cycle and it's child-cycle).

The forth strength of the TFM is the model-to-model transformations defined as part of TFM4MDA and improved in Uldis Doniņš work on TopUML. Thus, it is possible to define the domain model, identify the CIM (in case its scope is smaller than the domain model's) and execute a model transformation to PIM/PSM. Most of the transitions between TFM and UML diagrams can be automated, and domain experts can then validate and check the acquired diagrams [13]. The following UML diagrams are considered by the TopUML approach: 1) Use Case diagram; 2) Sequence diagram; 3) Activity diagram; 4) Interaction Overview diagram; 5) Communication diagram; 6) Class diagram; 7) Object diagram; 8) State diagram; 9) Package diagram; 10) Component diagram; 11) Deployment diagram.

2.1.3. Weaknesses of Topological Functioning Model

TFM is a powerful tool for domain modeling with some important strengths. However, TFM has also weaknesses that make this approach heavy to use, complex and unfamiliar to the software engineers. The underlying architecture of TFM is that functionality determines the structure of the planned system [65]. Researchers in a similar field have noticed the TFM approach and consider TFM to be a drive towards a CIM specification for MDA. However they conclude that some important pre-CIM and design considerations may have not be taken into account [21]. The first weakness of the TFM is that it does not provide all aspects of a domain model. From perspective of domain modeling TFM only describes the procedural knowledge and lacks the declarative knowledge. The distinction between these is going to be discussed in the next section – Knowledge Integration. TFM is strong at describing the business processes and rules (as conditions), but does not provide means to define the business structures directly or a vocabulary for the business system.

The second weakness of the TFM and particularly TFM4MDA is its beginning. This approach starts with an informal description – text in natural language. First of all, this description needs to be produced, which will become a part of the domain knowledge. Since effort and resources are going to be put into producing this informal

description, the question may arise – is it worth investing into such an informal and unstructured artifact? If one is going to capture domain knowledge in an artifact for software engineering or business analysis, he has more formal and structured options to choose than a free text. On the other hand, the author agrees that this artifact should not be something too complex since it is meant for the business people to understand as well. Nevertheless, at the moment the analysis of such an informal description for production of a formal business process would be too complex for the following reasons. There is no structure of the description to define the sequence and branching of events. To define conditions for events one would need to use sentence with "If" and if the same condition applies to several events, it somehow needs to be also in those representing sentences. Moreover, how to handle ambiguity of the text when the same thing is named differently? Another serious problem with an informal description is transparency. If a TFM is produced from this text, and a correction is necessary or a mistake is identified, then it is hard to trace anything back to the source description.

The third weakness is that there is no tool support for TFM, TFM4MDA or TopUML. As of now the only way to acquiring a TFM is by doing manual informal description development and analysis, then manual creation of the functional features, and definition of the cause/effect relationships. This is all done manually and with a high possibility of human error, because of lack of transparency.

The forth weakness is that although there are guides and examples available, this process is heavy and distant from standard software engineering approaches. Since domain model needs to be understood by the business people as well as by the technical people, TFM is too unfamiliar and complex to serve as the only model representing the domain.

2.1.4. Substantiation for Improvements

Based on the strengths of the TFM approaches it is clear to see that this is powerful approach that brings a lot of benefits to domain modeling. However, the drawbacks mentioned in previous section are preventing this approach from being

effectively used by system analysts other than the TFM research group. First of all, tool support is vital for any domain modeling approach today.

In order to develop a tool support for TFM4MDA as described until now, one would need to deal with very complex natural language processing and the informal description. Author has looked at some approaches that use NLP for domain modeling (e.g. LIDA and NIBA in section 1.2.), but these approaches provide limited results or are still based on some structure of the text. For the purpose of acquiring TFM this would be a very complex task and there would need to be rules and structure defined for the informal description anyway. This would complicate the whole TFM construction process even more. And one should not forget the problem of transparency and tracing the elements to the source. Thus, one of the necessary improvements is an introduction of a more formal (structured) way to describe the business system functioning than an informal description. Also to mitigate the lack of a possibility to model business structure and vocabulary with the TFM, there should be a possibility to integrate with other approaches, which can provide the necessary facility and information. Furthermore, in research [5] authors state that transformation from the CIM-Knowledge Model to the CIM-Business Model cannot be performed automatically, because it requires human participation since information in the CIM-Knowledge Model is specified informally and used to have implicit knowledge. Thus, another necessary improvement is to acquire a formal CIM-Knowledge Model and enable the possibility to transform this model to the CIM-Business Model with a formal model transformation (automatically).

2.2. Innovation of the Integrated Domain Modeling

Computer science has come a far way in understanding knowledge that exists in the real world and developing means to capture and manage this knowledge. Knowledge engineers are mostly associated with artificial intelligence (AI), but many aspects of system analysis also deal with knowledge. There have been significant results and applications in both artificial intelligence (AI) and system analysis. On the other hand, the integration between these two domains and the benefits it can offer has not yet been fully recognized. There are few approaches that have been going in this

direction, however none of these approaches suggest a solution for acquiring the domain model automatically from the corresponding domain knowledge, which should be the case. There is no reason why we could not automatically generate a model for a domain, for which we have all the corresponding knowledge explicitly defined.

Other authors have been investigating how to combine AI and system analysis for the benefit of domain modeling, incorporating Ontologies with MDA. Ontologies, as formal representations of domain knowledge, enable knowledge sharing between different knowledge-based applications. Diverse techniques originating from the field of artificial intelligence are aimed at facilitating Ontology development. However, these techniques, although well known to AI experts, are typically unknown to a large population of software engineers [25]. In order to overcome the gap between the knowledge of software engineering practitioners and AI techniques, a few proposals have been made suggesting the use of well-known software engineering techniques, such as UML, for Ontology development. An approach proposed in [11] is dealing with generating Resource Description Framework (RDF) from a UML model. RDF is a W3C XML-based standard for sharing Ontologies on the Semantic Web. Another approach [19] proposes a transformation to semantic extraction of Ontologies from UML models. Their initial presumption is that UML and Ontologies complement each other. That is to say, UML is designed for building models by human experts, while OWL is designed to be used at run time by intelligent processing methods. Another similar approach for integrating Ontologies with MDA is suggested in [9]. These approaches mentioned earlier are useful if you already have the design model (UML) and want to acquire the Ontology. From the perspective of MDA the order of the acquired artifacts is incorrect, because the design model or PIM/PSM should be derived from CIM, which includes the declarative knowledge provided by Ontology. This means that the Ontology has to come first, in order to construct an accurate PIM/PSM. The IDM insists on starting with knowledge and not the design.

To step towards the completeness of MDA and enable the automation of system analysis and software development the IDM approach is proposed. The IDM approach provides a formal way to facilitate Ontology for software engineering. It does not suggest a novel methodology for Ontology development, but instead is based on the existing methodologies. This approach suggests Ontology to be directly used as an

input for domain modeling by exploiting business Use Cases and natural language processing (NLP). Thus combining the declarative and procedural knowledge for the domain modelling. The IDM approach shows how declarative and procedural knowledge complement each other and can be compared for validation purposes. There are several approaches for domain modeling, which also have a supporting toolset, but usually they require the users to learn a new form of domain knowledge accepted by the approach and the tools, and then manually construct the domain model. The IDM approach, on the other hand, integrates the common standards used in business environment – Use Cases and Ontology, and then provides the means to generate the initial domain model automatically. This level of automation and reuse of enterprise standards is the main innovation of the IDM approach.

2.3. Knowledge Integration

This sub-section discusses the concept of declarative and procedural knowledge and shows why both are integral parts of the domain knowledge. Moreover, the author proposes to connect both parts to construct a domain model to be used within MDA. Thus the IDM integrates the declarative and procedural knowledge for domain modeling. This sub-section also gives an outline of the IDM approach and discusses the different components and dependencies between them. Here the author shows a high level composition of the IDM approach and explains the different activities involved.

2.3.1. Distinction Between Declarative and Procedural Knowledge

At the beginning of domain modeling it is necessary to acquire knowledge about the domain – actual business system and its environment. By business system here the author means a real world entity functioning with its business processes within its environment. In software engineering the system analysis is interested in knowledge that can be articulated or captured in form of text, tables, diagrams, product specifications and so on. Knowledge means understanding of a subject area including concepts and facts about that subject area, as well as relations among them and

mechanisms for how to combine them to solve problems in that area [25]. The traditional Artificial Intelligence (AI) techniques most frequently used to represent knowledge in practical intelligent systems include object-attribute-value triplets, uncertain facts, fuzzy facts, rules, semantic networks, and frames; Ontologies have acquired major importance in knowledge representation as well [25]. On the other hand, software engineers have developed means to manage knowledge about business systems and processes, e.g. Business Process Modeling Notation (BPMN) and Model Driven Architecture (MDA) discussed in previous sections.

The domain knowledge can be considered as part of the CIM, i.e., Knowledge Model [5]. The term knowledge can be used to refer to a state of knowing facts, methods, principles, techniques and so on. This common usage corresponds to what is often referred to as "know about". Second, usage of the term knowledge is when it refers to understanding facts, methods, principles and techniques sufficient to apply them in the course of making things happen. This corresponds to "know how". Cognitive psychologists sort knowledge into two categories: declarative and procedural [73]. From the perspective of a student who is learning knowledge: 1) Declarative knowledge is that the student knows or understands. For example, "Riga is the capital of Latvia" or "Book catalogue has entries of books"; 2) Procedural knowledge is that the student is able to do something. For example, he knows how to buy an airplane ticket to Riga or get a book at the library. This distinction between declarative and procedural knowledge may seem obvious, but has not been recognized too often in computer science except knowledge engineering, which is part of AI. Knowledge engineer must have clear understanding of the representation scheme so that problem can be easily handled. Knowledge can be declarative or procedural [34]. Declarative knowledge is a sentence depicting facts while procedural knowledge is a group of sentences, which can be carried in a series of steps by building relationships. Knowledge must be encoded either as a part of expert systems or should be easily available to them. In this research [34] the authors use rules, semantic nets, frames and Ontology as their domain knowledge representations.

2.3.2. Outline of the Integrated Domain Modeling Approach

The domain knowledge about the business system and its environment usually exists in different documents in a form of natural language. It is necessary to store this knowledge in a way, so that a computer could understand it (there has to be more structure than natural language can provide). To properly describe a business system in its environment, it is necessary to have both – declarative and procedural knowledge. Approaches like BPMN are very strong describing the procedural knowledge, but lack the AI strength in dealing with declarative knowledge. The IDM approach integrates declarative and procedural knowledge providing a common approach for system analysis with the perspective of integrating with MDA.

Figure 2.1 the author shows how declarative and procedural knowledge can be integrated to acquire a CIM for MDA. It starts with the domain knowledge defined as formal CIM-Knowledge Model, which includes business Use Cases and Ontology. It then provides a formal transformation to CIM-Business Model, which is defined by TFM. The next step is acquiring the Business Model. When Knowledge Model is constructed and verified, it is possible to generate the Business Model automatically using the TFM generation algorithm described in [83]. This algorithm utilizes the statistical parser to analyze the syntax of Use Case sentences and identify functional features for the TFM. Nevertheless, TFM will have to be validated as well. If any changes are necessary, they will have to be done in the Knowledge Model and then the TFM can be regenerated. After that according to MDA it is possible to transform the CIM to PIM/PSM. The source for this transformation is the Business Model (CIM) and the target is the Design Model (PIM/PSM). Detailed transformation to UML from TFM is described in U. Doniņš doctoral thesis [13].

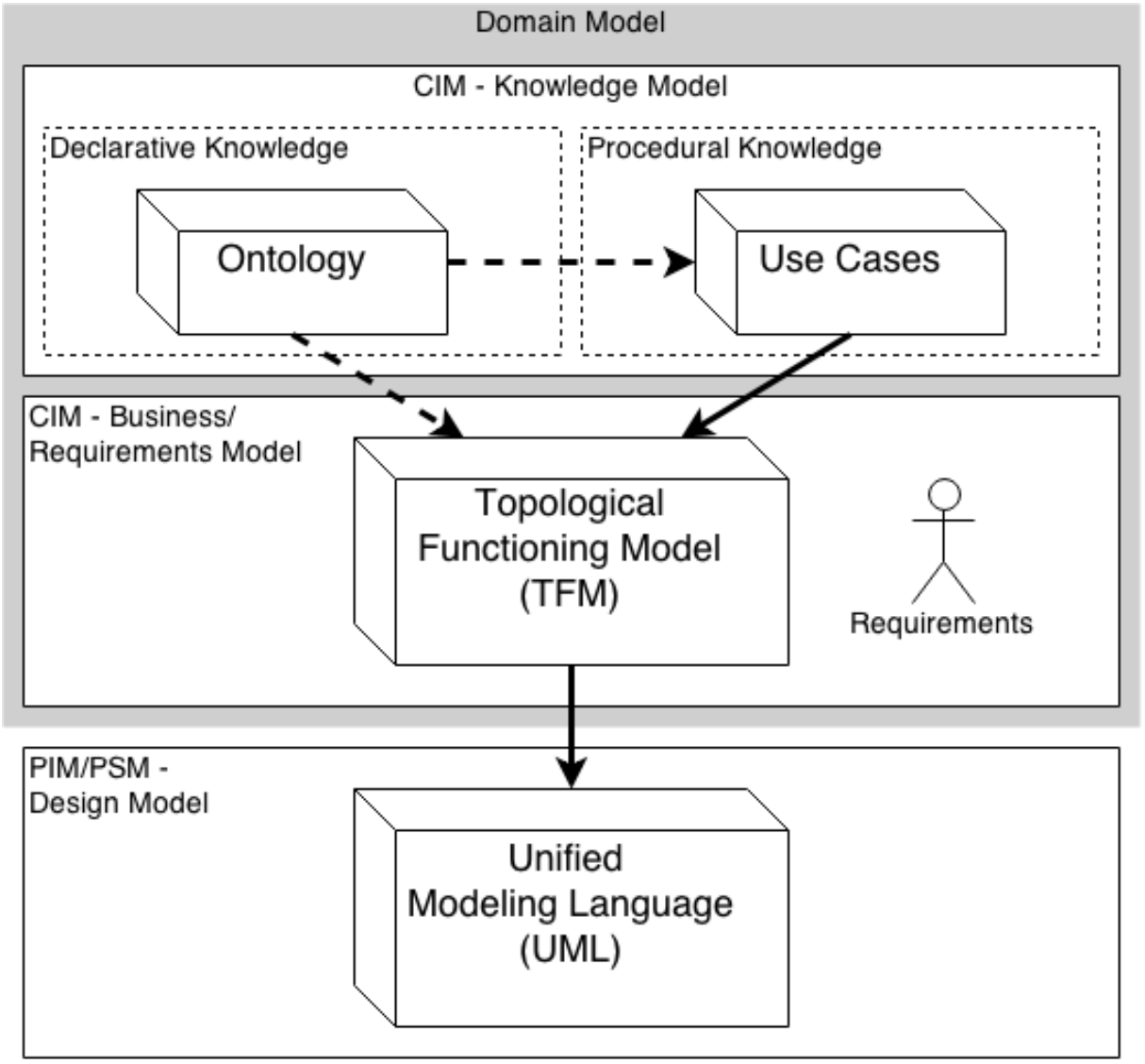


Figure 2.1. Outline of the IDM approach

2.3.3. Acquiring Declarative Knowledge by Means of Ontology

This sub-section shows in detail how to acquire the declarative knowledge of a business domain, and how to formally define this knowledge by means of Ontology. The Ontology standard OWL is reused for the IDM approach. An example of developing Ontology for a library domain is provided to showcase the IDM approach in action. Ontology is a perfect candidate for representing declarative knowledge about a business system and its environment. Ontology is an extremely important part of the knowledge about any domain. Moreover, Ontology is the fundamental part of the knowledge, and all other knowledge should rely on it and refer to it. In Figure 2.2 the

author gives example Ontology for a library domain - the class hierarchy shown. It includes the main classes that a library business system needs to function.

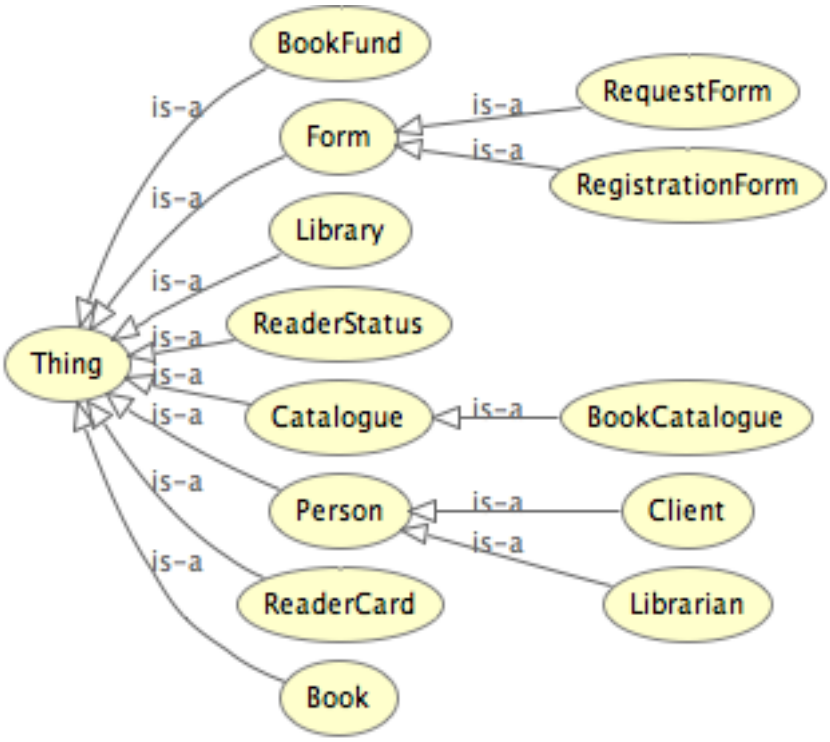


Figure 2.2. Ontology class hierarchy for a library domain

The main actors are Client and Librarian, which both are sub-classes of Person. Class hierarchy includes also important concepts for a Library business system like Library, Book, Book Catalogue, Reader Card and Request Form. The author used Protégé [75] to develop the Ontology.

Figure 2.3 shows the description of the librarian class. It shows some of the properties and relationships between properties and classes. For example, there is an object property for the “Librarian” class “checkOut”, which states that a librarian would checkout a book, not any other object. Another example, object property “hand” states that a librarian can hand out a book, a registration form and a request form. In a similar way properties need to be set for all classes, if it is in the scope of the domain.

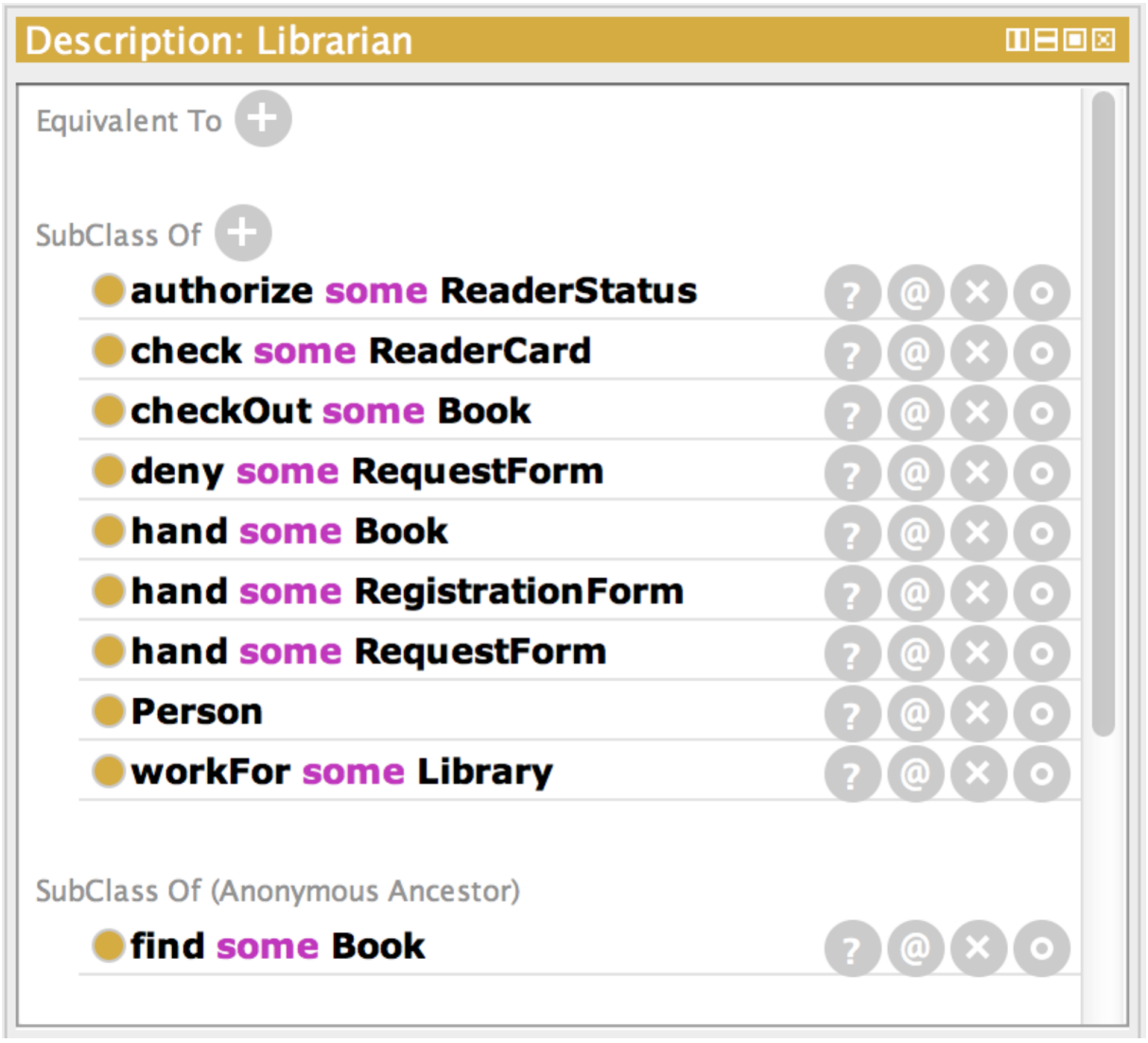


Figure 2.3. Ontology class properties for a library domain

Developing an Ontology includes: 1) defining classes in the Ontology; 2) arranging the classes in a taxonomic hierarchy; 3) defining properties and describing the relationships with classes; 4) defining the individuals. Creating the class hierarchy is the first and second step. The third step is defining the properties and relationships between classes and properties. It is important that all the business system's concepts and actions that can be associated with these concepts are defined. Recognizing a satisfactory scope for the domain is not easy. It will not always be possible to capture everything on the first try, but as mentioned before this has to be an iterative process. The ability to modify the Ontology is crucial, because the scope of the domain can also change. The author does not propose a methodology for developing Ontologies; instead to use an already developed Ontology for further knowledge engineering and system analysis, if one exists in the company. If there is no Ontology defined for the

business system already, then it is necessary to build a new Ontology by analyzing the business system and its environment. Available documents and expert knowledge is the main input for this development. Particularly in the case of information extraction almost every text introduces new important terms, so one should not assume that all terms encountered in the text will already be included in an existing Ontology. The ability to add new terms to an existing Ontology is crucial even when using an Ontology whose structure has been formally defined [28]. This means that even if we initially have a defined Ontology, it might lack some classes and properties for the business system or its environments under consideration.

2.3.4. Acquiring Procedural Knowledge by Means of Use Cases

This sub-section shows in detail how to acquire the procedural knowledge of a business domain, and how to formally define this knowledge by the means of Use Cases. A special form of Use Cases (meta-model) is required and developed for the IDM approach. A meta-model of these Use Cases is discussed and how it can be used to define procedural knowledge. An example of developing Use Cases for a library domain is provided to showcase the IDM approach in action.

Business Use Cases or Use Cases is a common approach widely used in software engineering. Use cases should not be confused with UML Use Case diagram. Use cases are not normalized or standardized by any consortium, unlike UML Use Case diagram, which is defined by OMG. In fact, there are many different Use Case templates and the structure of a Use Case can be adjusted depending on the situation and the development team [39]. These textual Use Cases will be used for representing the procedural knowledge within IDM approach. The following structure of a Use Case is going to be used: 1) Use Case description; 2) actors; 3) main scenario; 4) extensions; 5) sub-variations. Each step has a description, pre-condition, and post-condition. The structure in detail will be discussed in section 3, when author develops a supporting toolset for IDM.

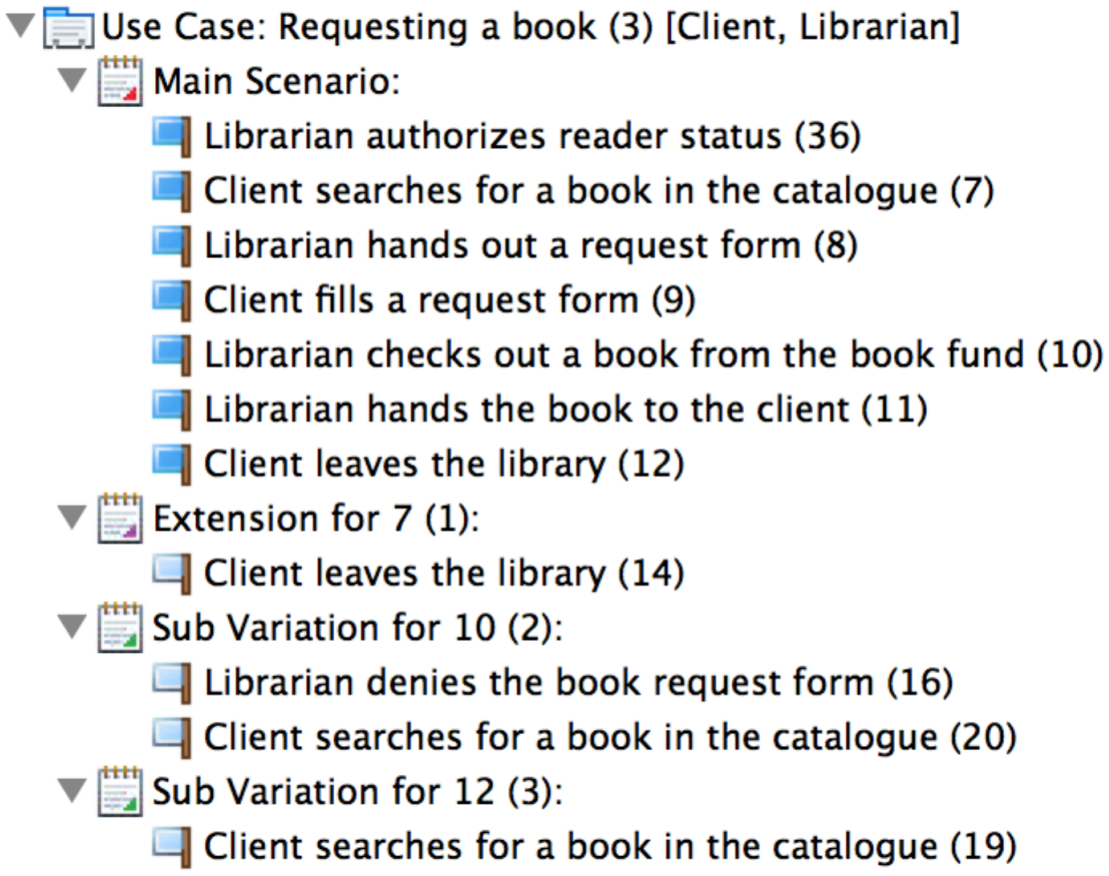


Figure 2.4. Sample Use Case for a library domain – requesting a book

Figure 2.4 shows a Use Case for requesting a book in a library. First a client searches for a book in the catalogue (step 7) and then fills a request form to get the book (step 9). Librarian's responsibility is to hand the request form (step 8), check out the book from the book fund (step 10) and hand the book to the client (step 11). There are also alternative scenarios – when the client leaves without requesting a book (extension 1), when librarian denies a book request (sub-variations 2), and when client wants to get another book (sub-variation 3).

Ontologies, discussed in the previous section, provide logical statements that describe what terms are, how they are related to each other, and how they can or cannot be related to each other. Use Cases on the other hand provide a formal way to define the procedural knowledge, showing step by step how a process is executed, what the variations are and which actors are involved. The problem with Use Cases is that their steps are sentences in natural language. Use cases cannot guarantee that the terms used in the step sentences will be unambiguous. For example, there could be steps "Client fills a request form" and "Librarian denies a form". These steps are correct from a syntax perspective, but they are ambiguous, because in the first

sentence a form that is meant for requesting a book is defined as “Request form”, but in the second sentence it is defined as “Form”. This would not be a problem if there were a predefined vocabulary, which determines that these terms mean the same thing in this domain. Another problem is potential inconsistency to the domain. For example, there could be a step “Librarian shows a reader card”. This step is perfectly correct from a perspective of syntax and may seem to make sense, because librarian can also be a reader in a library, but for the given domain “Librarian” is a definition of the person who works for the library and at this moment in time is fulfilling this role. So in fact this sentence does not make sense from the perspective of the domain.

Some more examples from the library business system can be seen below in Figures 2.5 and 2.6.

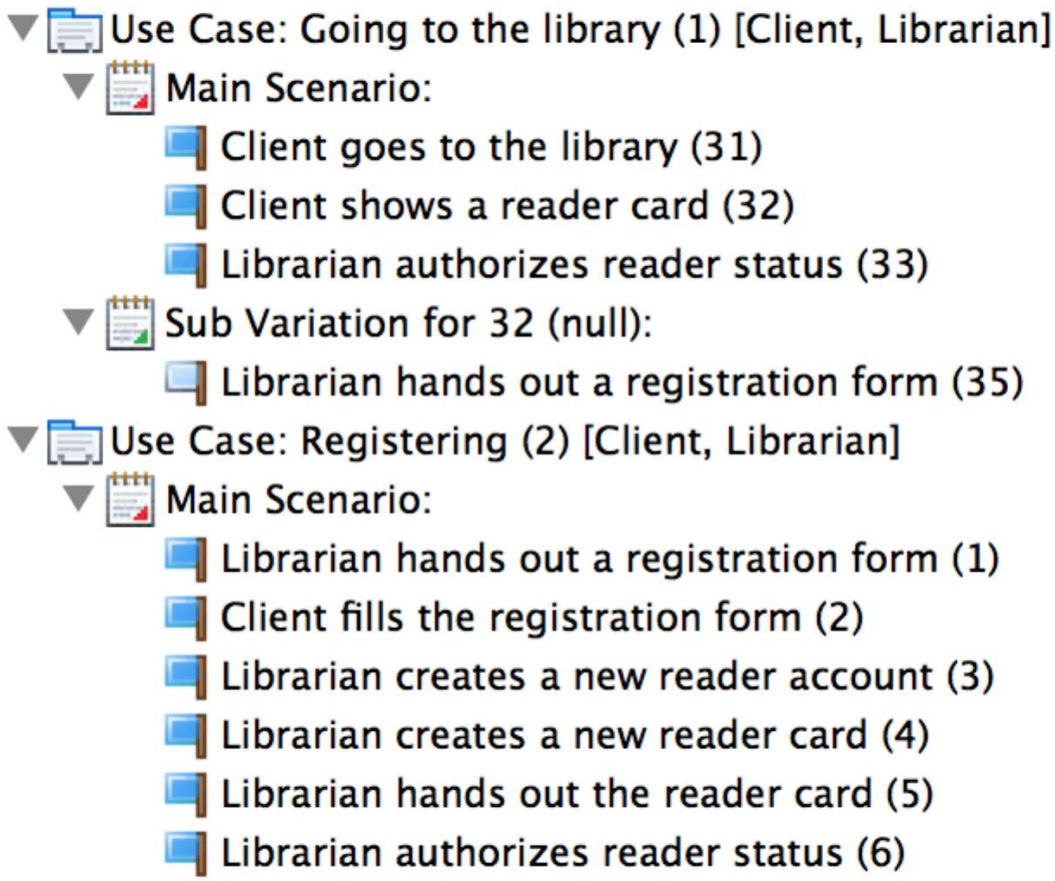


Figure 2.5. Sample Use Cases for a library domain – going to the library, registering

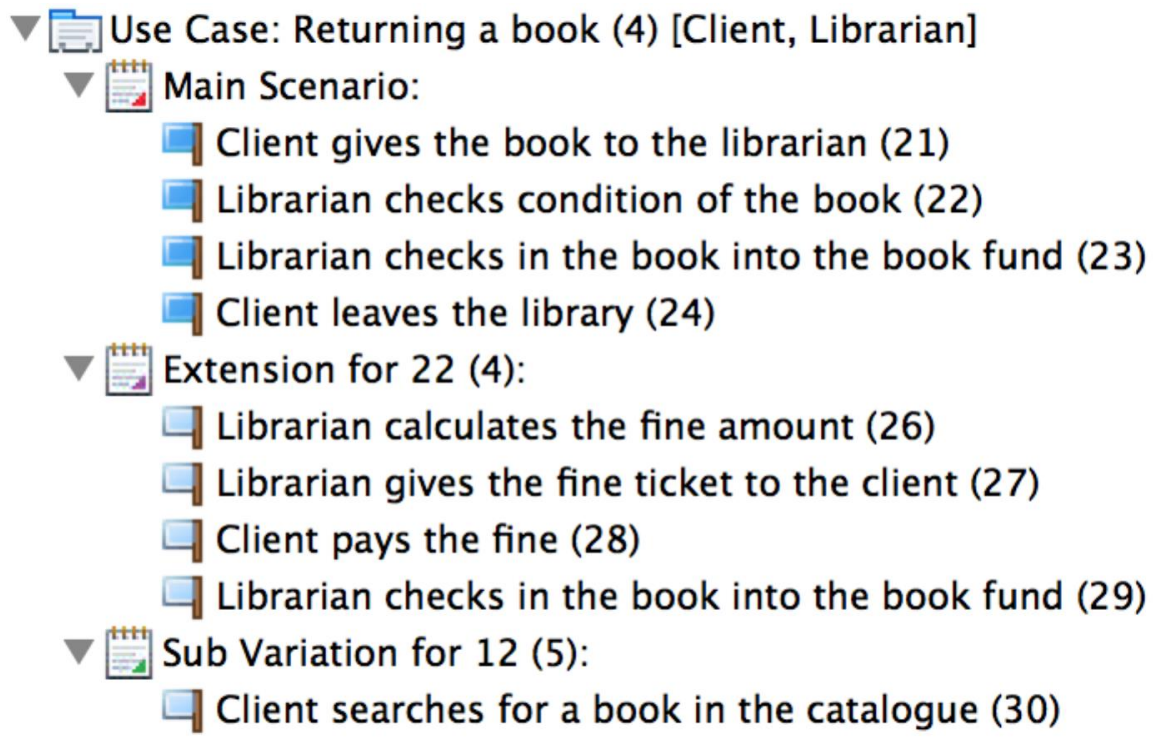


Figure 2.6. Sample Use Case for a library domain – returning a book

2.4. Natural Language Processing

This sub-section demonstrates how Natural Language Processing (NLP) is applied within IDM. NLP is not the main focus of this research, but since the author deals with domain knowledge expressed in natural language, NLP is an important tool to make sense of the knowledge. NLP enables the knowledge to be understood by the machine and thus allow for validation and transformation possibility. IDM validates Use Cases model against the Ontology of the business domain to ensure the unambiguity of Use Case steps as well as the correspondence to the business domain. An example of validating Use Cases of a library domain is provided to showcase the IDM approach in action.

A statistical parser can be used for analyzing the sentences of Use Cases, and thus retrieving data for functional features to construct the TFM of the business system (more details on this will be discussed in next sub-section). Fetching Functional Features from Use Cases is a straightforward task as long as the sentences of the Use Cases are kept as simple as possible and in simple present tense. It is the task of system analyst to prevent incompleteness, ambiguity or inconsistency of Use Case

sentences. However, it would also be possible to address this kind of ambiguity and inconsistency with NLP and Ontology, and provide some level of automation.

For the first problem of ambiguity, consider the first step “Client searches for a book in a catalogue” (see Figure 2.4). It is possible to rephrase this “Client searches for a book in a book catalogue”. Notice that the second sentence specifies that it is a book catalogue and not just any catalogue. In this specific domain both sentences refer to the same object and it is important, that when the steps get analyzed the correct objects are considered. Consider the library Ontology’s class hierarchy (see Figure 2.2), it is clearly defined that “Catalogue” is a super-class of “Book catalogue”. This solves the problem of ambiguity in this case, because it refers to the same object. In case there would be more than on sub-class for “Catalogue” it would be possible to automatically suggest choosing one of the sub-classes. Moreover, Ontology provides the possibility to define synonyms with the property “EquivalentTo”. For example, if in the business environment the “Book Catalogue” is usually referred to as “Directory”, it would be possible to define “Directory” as a class and add it as “EquivalentTo” to “Book Catalogue”.

Another problem is the problem of being inconsistent to the domain. When rephrasing the same step like this “Client searches for literature in a catalogue”. The concept “Literature” is not defined in the Ontology, so this sentence should be marked as invalid until someone defines the concept in the Ontology. The same applies to the properties. If the sentence is “Client looks for a book in a catalogue” and property “look” is not defined, this step should be marked as invalid. Another example sentence – “Librarian shows a reader card”. From Ontology’s properties (see Figure 2.3) and class relationships (see Figure 2.2) it is possible to see that the relationship between “Librarian” and “reader card” is defined by property “check” and not “show”. This implies that the sentence is invalid and should be corrected or the Ontology has to be modified. By checking the correspondence between properties and classes it is possible to deal with this problem.

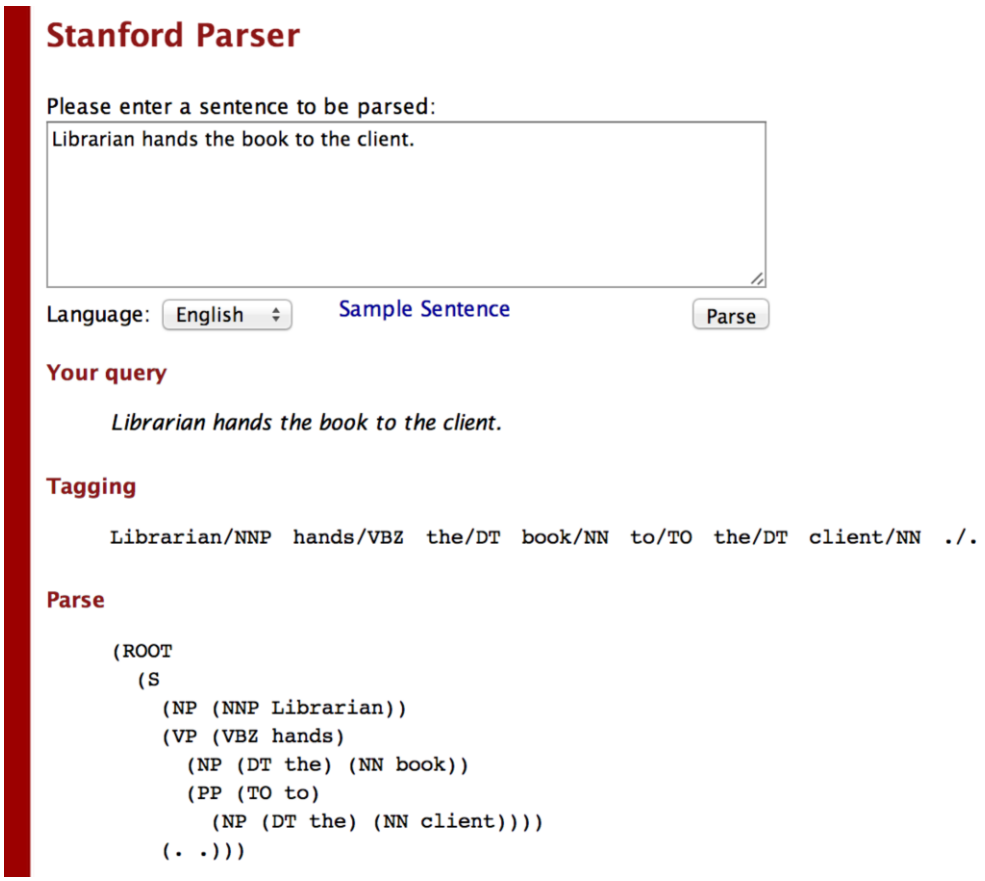


Figure 2.7. Sample parse tree

This is a parse tree generated by Stanford Statistical Parser [88] from a business Use Case step sentence “Librarian hands the book to the client”. The parse tree is represented by part-of-speech tags [1]: S – sentence; NP – noun phrase; VP – verb phrase; NNP – proper noun; VBZ – verb, 3rd person singular present; DT – determiner; TO - to.

To implement the solution for these ambiguity and inconsistency problems technically it is possible to analyze the parse trees of the sentences. Figure 2.7 shows a parse tree for Use Case step “Librarian hands the book to a client”. This sentence can be broken down into verb phrase, noun phrase, and then also into verbs and nouns. Approach for knowledge integration suggests that the nouns need to correspond to the classes and the verbs need to correspond to the properties. If they do not, then an error should be raised and either the Ontology or the business Use Case needs to be modified. For this example we see that “Librarian”, ”Book”, “Client” and “hand” is defined by the library Ontology, so we have a valid sentence from perspective of ambiguity – all terms are defined and understandable. From perspective of consistency

"Librarian" and "Book" are associated with "hand". This association can be confirmed by the Ontology, because a librarian hands a book.

2.5. Generating a Computation Independent Model via Model Transformation

This sub-section introduces the principles of Use Cases model transformation to TFM model and also the use of NLP for identifying the corresponding components for the model-to-model transformation. For model transformation NLP is used to identify the actor or entity responsible and the description of the functional feature. An example is provided of doing the transformation from a Use Cases for a library domain to a corresponding TFM to demonstrate the IDM approach in action.

At this point of the IDM's life cycle there is an Ontology and Use Cases defined for the business domain. The next goal is to acquire the TFM. For this 2 main steps are necessary: 1) retrieve functional features; 2) retrieve topology.

2.5.1. Retrieving Functional Features

Functional features are represented by a tuple consisting of action, a result of this action, an object involved in this action, a set of preconditions of this action, an entity responsible for this action, and subordination [62]. Object's action, a result of this action and an object involved in this action are merged into functional feature description attribute. Because of the complexity of text analysis this is left for future work. However, the entity responsible and the description will be fetched using NLP.

Use cases are formed by sentences written in natural language. Every sentence, except title and actors, in a Use Case can be considered as a representation of a functional feature. Use case sentence can sometimes represent more than one functional feature. This can happen when sentence consists of more than one result of the action or objects involved in the action. Such an issue can be dealt with by analyzing sentence's coordinating conjunctions. For example in Figure 2.4, if 2nd and 3rd steps of Use Case "Requesting a book" are combined in one sentence "Librarian hands out a request form and client fills the request form". In this sentence the second

reference to a request form could be replaced by a pronoun “it”. This should be taken into account, but it is recommended not to use complex sentences within IDM approach. So in this case, one should write down 2 Use Case steps instead of 1 complex. Moreover, the sentences of a Use Case should be written as simple and unambiguous as possible, but in realistic case this is not always possible. In the examples used in this work Use Case step sentences are constructed to answer this question – who does what? For example, “Librarian checks out the book from book fund”. The verb phrase of the Use Case step’s sentence is considered the action. Moreover, Use Case’s actors will be considered as objects involved in the action and entities responsible for the action. The title can partly be considered as a functional requirement.

IDM has to be able to form the corresponding functional features by analyzing the Use Case sentences. For this purpose natural language processing methods have to be applied. Concrete syntax tree, or parse tree for short, will be used for the analysis of Use Case sentences. Parse tree is a tree that represents the syntactic structure of a sentence according to some formal grammar [31]. Parse trees are usually output of parsers, which can use different methods for finding the right parse tree for the specific sentence. The most efficient parsers are statistical parsers, which associate grammar rules with probability. For example, Use Case sentences “Librarian checks out the book from book fund” and “Librarian creates a new reader account” will be parsed using The Stanford Parser [21]; results are shown in figure 2.8 By exploiting statistical parser it is possible to acquire the structure of the sentence, and thus analyze it.

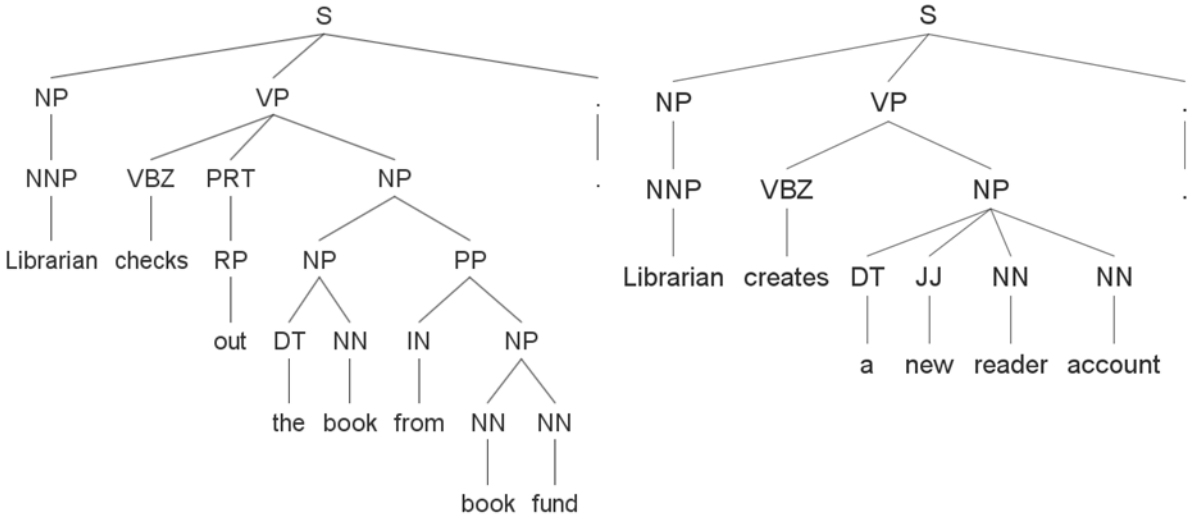


Figure 2.8. Sample parse trees for functional feature retrieval

Figure 2.8 shows sample parse trees of Use Case sentences. The abbreviations are part-of-speech tags according to [76]: S – sentence, NP – noun phrase, VP – verb phrase, NNP – proper noun, singular, VBZ – verb, 3rd person singular present, DT – determinant (article), NN – noun singular or mass, PP – prepositional phrase, TO – "to", PRT – particle phrase, RP – particle, IN – subordinating conjunction, JJ – adjective. In the first sentence in Figure 2.8 an action of the corresponding functional feature has to be identified. In this case it is the verb phrase (VP tag) of the sentence – "checks out the book from book fund". It consists of the object action (checks), the result of the action (book) and object involved in the action (book fund). The responsible entity for the action can be determined by comparing the actor list of the Use Case and the noun phrase (NP tag). In this case the noun phrase is "Librarian" and there is "Librarian" in the actors list as well, so the entity responsible for the action probably is "Librarian". Preconditions can be taken from the Use Case step preconditions directly. However, only the system analyst can determine functional feature's subordination after acquiring the TFM (subordination can be inner or outer, depending whether the functional feature is an inner part of the business system or an input/output to surrounding environment).

By analyzing Use Case sentences it should be possible to derive functional features. It is important that during this analysis functional features represented by the same tuple are considered the same functional feature. This means that no duplicate functional features are created and two or more Use Cases can include the representation of the same functional features.

2.5.2. Retrieving Topology

Once there is a set of functional features it is necessary to retrieve the topology of TFM. Topology means that cause-effect relations between functional features need to be identified and created. The structure of the Use Cases will help with this task.

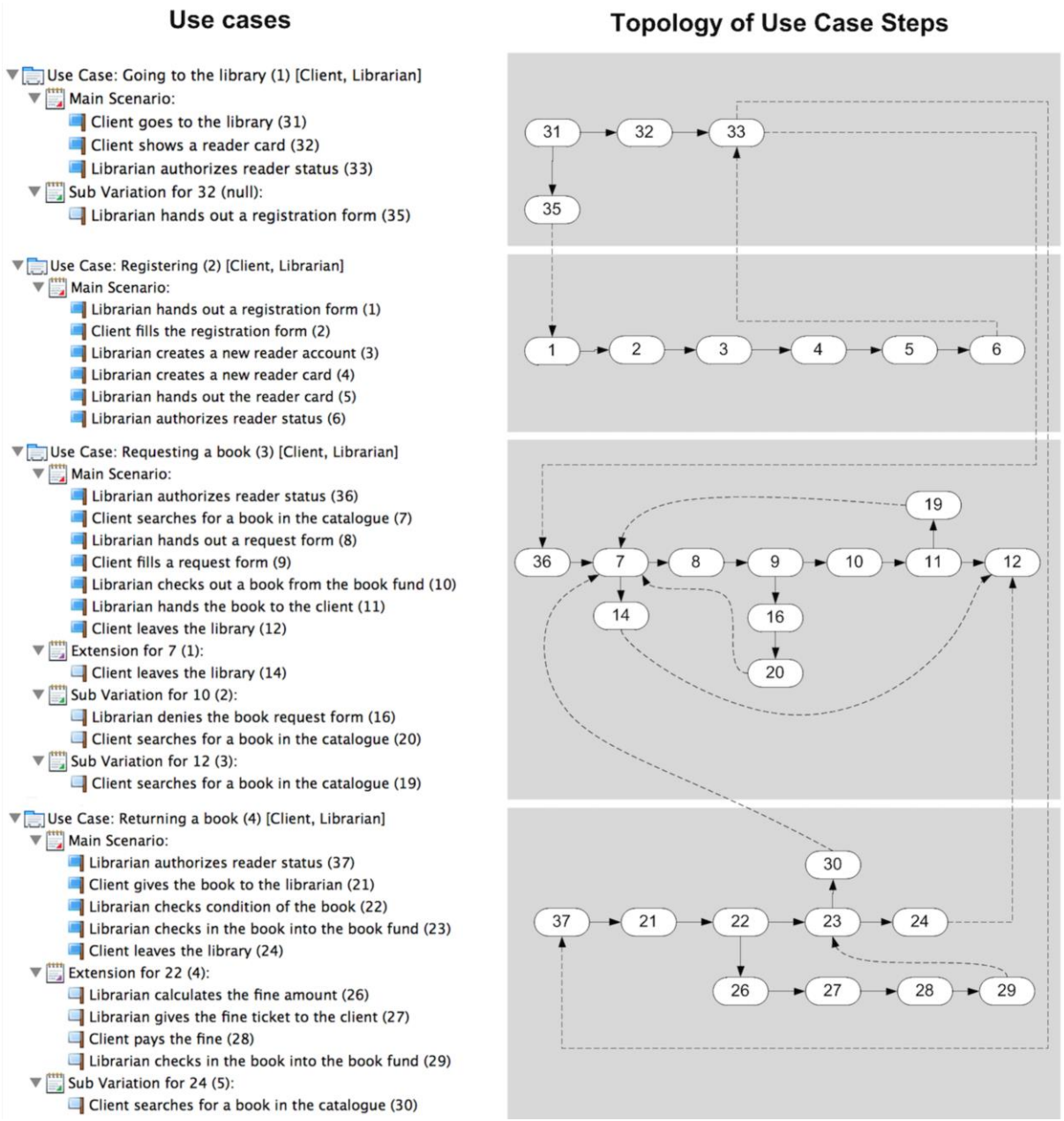


Figure 2.9. Topology of Use Case Steps

First of all, every Use Case's main scenario is an ordered sequence of functional features. Additionally, by analyzing the extensions and sub-variations it is possible to detect branching in a TFM. Extension adds an effect to the functional feature represented by the step referenced by the extension. However, sub-variation adds an effect to the functional feature represented by the previous step referenced by the sub-variation. Therefore, the setting of cause-effect relations between functional features represented within the same Use Case is very straightforward. As shown in Figure 2.9 the 4 main sequences of functional features come from main scenarios of Use Cases. The identifiers of the Use Case steps are reused for the Functional

Features, so for example Use Case step “31” corresponds to Functional Feature “31”. As a demonstration of Use Case extension and sub-variation analysis consider functional features number 1 and 11. Functional feature number 1 has an additional effect because of the sub-variation 2a, but functional feature 11 has an additional effect because of the extension 4a.

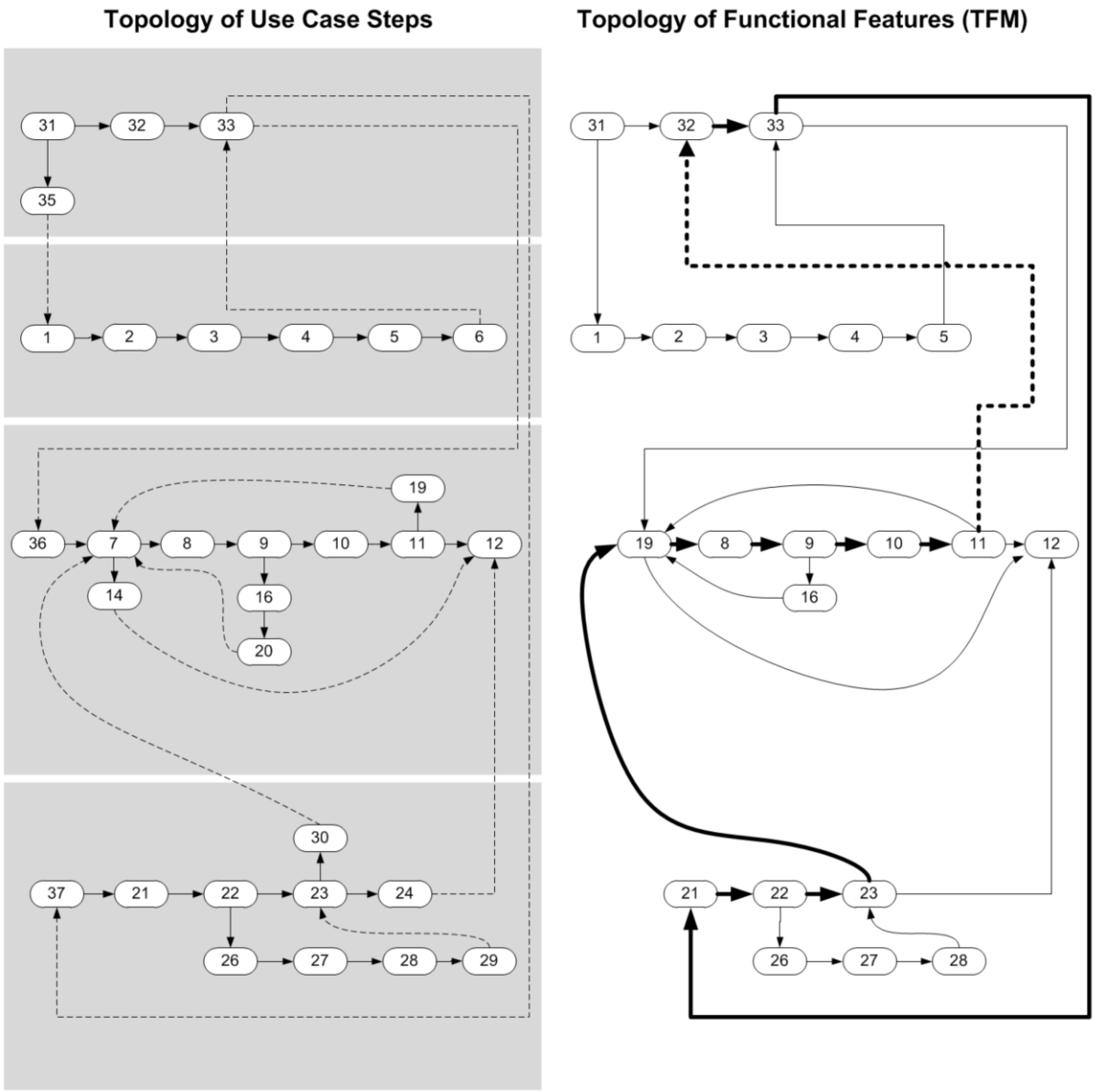


Figure 2.10. Acquiring Topological Relationships from Use Cases

Figure 2.10 shows the topology Use Case steps in compliance with cause-effect relations between functional features. IDM suggests several iterations back and forth between Use Cases and TFM until the system analyst can verify it is correct. As part of the verification the analyst could present the acquired TFM (a graphical view of the

previously written process) to the stakeholders for approval. The mapping between Use Case sentences, functional features and TFM should be intuitively illustrated and easily editable, so that any incompleteness, redundancy or inconsistency could be corrected. The bold arrows represent the main functioning cycle of the TFM. The functional features are set to be part of the main functioning cycle by the system analyst manually. By analyzing the Use Cases it is possible to derive a TFM.

For the example shown in figure 2.10, the main functioning cycle consists of these functional features in the following sequence (represented by the identifier and corresponding Use Case step description):

- 32 – Client shows a reader card;
- 33 – Librarian authorizes reader status;
- 21 – Client gives a book to the Librarian;
- 22 – Librarian checks the condition of the book;
- 19 – Client searches for a book in the catalogue;
- 8 – Librarian hands out a request form;
- 9 – Client fills a request form;
- 10 – Librarian checks out a book from the book fund;
- 11 – Librarian hands the book to the client.

The main functioning cycle includes all main processes of the example library system starting from client coming to the library, client returning a book and client requesting a book. There are also other cycles in this TFM including the registration of a client or requesting a book without returning one first. The main functioning cycle must be defined and set by the analyst based on input from domain experts. In the library example, the system analyst for the completeness of main functioning cycle sets cause-effect relation between functional features with identifier 11 and 32 as shown in Figure 2.10 with a dotted arrow. It cannot be determined automatically.

Another task is setting the cause-effect relations between functional features fetched from different Use Cases. Precondition section of Use Cases are used to define this relation, because it contains the Use Case step, which is the cause of the particular functional feature. For example, Use Case's "Requesting a book" precondition is "Librarian authorizes reader status", which is the 3rd step of Use Case's "Arriving" main scenario. Moreover, as different Use Case sentences represent the same

functional feature if their tuples conform, relation between different Use Cases can be fetched from extensions and sub-variations, too.

2.5.3. Use Cases to TFM Transformation

Here the author is going to describe the algorithm for transforming Use Cases to TFM. First some attributes will be introduced: U – a set of Use Cases describing the business system, A – a set of actors for a Use Case, P – a set of pre-conditions, M – a set of steps of the main scenario for a Use Case, E – a set of alternative scenarios (extension or sub-variation) for a Use Case, S - , F – set of functional features for a TFM.

1. Assign the next unprocessed Use Case to U;
2. Assign the next unprocessed step of the main scenario of U to M;
3. Create a new functional feature and add to set F with the following tuple, if there is no existing functional feature with the same tuple:
    a. Description – the verb phrase of description of M;
    b. Pre-condition – append pre-condition of M;
    c. Post-condition – append post-condition of M;
    d. Responsible entity – if description of M noun phrase matches with one of the actors of U, then this actor (otherwise there should be an error returned).
4. If there is an existing functional feature with the same tuple, then append the pre-condition and post-conditions of the step to the existing one and also create the corresponding topological relationships to the existing functional feature (treat this step as duplicate and merge it's properties);
5. If set F has at least 1 functional feature, add the next functional feature from M as the effect of the previous one;
6. Go to step 2 until there are no unprocessed steps in the main scenario;
7. Assign the next unprocessed extension or sub-variation to E;
8. Assign the next unprocessed alternative scenario step to S;
9. Create a new functional feature and assign it to F with the following tuple, if there is no existing functional feature with the same tuple:
    a. Description – the verb phrase of description of S;
    b. Pre-condition – append pre-condition of S;
    c. Post-condition – append post-condition of S;

d. Responsible entity – if description of S noun phrase matches with one of the actors of U, then this actor (otherwise there should be an error returned).

10. If there is an existing functional feature with the same tuple, then append the pre-condition and post-conditions of the step to the existing one and also create the corresponding topological relationships to the existing functional feature (treat this step as duplicate and merge it's properties);
11. For the first step of the alternative scenario - if an extension is processed in step 7, then set an effect for the extensions step's corresponding functional feature from mean scenario to the created functional feature;
12. For the first step of the alternative scenario - if a sub-variation is processed in step 7, then set an effect for the extensions step's previous corresponding functional feature from main scenario to the created functional feature;
13. If not the first step of the alternative scenario, then set the effect to the previous step's representing functional feature to the created functional feature of step 8;
14. Go to step 7 until there are no unprocessed steps in the alternative scenario;
15. Go to step 6 until there are no unprocessed alternative scenarios;
16. Go to step 1 until there are no unprocessed Use Cases;

The algorithm ends when all Use Cases are processed. The functional features and their topological relationships that have been identified are stored in variable F. This algorithm is going to be implemented with a model-to-model transformation according to MDA standards in section 3.

2.6. Summary

As shown in this section and also in author's publication [83] it is possible to automatically acquire TFM from business Use Cases using Use Cases to TFM transformation, which is part of the IDM approach. Nevertheless, creating these business Use Cases still require human participation – defining and analyzing the knowledge about the domain. According to [5] the input model for transformation to PIM/PSM is the Business Model, which in IDM is represented by TFM. Knowledge Model and Business Requirements are the source models for the Business Model, which in IDM are represented by a combination of Use Cases and Ontology.

The IDM Approach can support various software development methods since Use Cases are widely accepted and used by the software engineering community as the starting point of software development. Moreover, automatically acquiring a graphical representation of the domain and the possibility to do further model transformations can be very useful. On the other hand, Ontology would be appreciated by more precise and sophisticated methods, when mistakes in the design phase are costly and developments should not start before the design is validated. Nevertheless, in the IDM approach Ontology is recommended, but not mandatory. This means that it is also possible to only create Use Cases, generate the initial TFM and continue with CIM developments until you are satisfied with the result. This approach would be more suited for agile software development methods, because it requires less work on documentation (defining the Ontology) and relies on the teams experience to have errors (ambiguities or inconsistencies) in the Use Cases.

However, today there is no use of an approach without a supporting toolset. This model-to-model transformation to acquire a domain model was specifically developed by the author with the possibility for automation in mind. Since it was shown here how the transformation could be developed, all pre-requisites for developing a supporting toolset for the IDM approach are in place.

3. SUPPORTING TOOLSET AND APPLICATION

The Integrated Domain Modeling approach provides an elegant solution to the complexity of domain model construction. However, for this to work in a real business environment for a business system modeling case there have to be tools to support the IDM approach. Moreover, these tools need to correspond to MDA standards and be based on available MDA frameworks. This will ensure that the toolset is extendable and can be integrated with other modeling tools, thus becoming a part of the MDA life cycle. This section introduces the supporting toolset for the IDM approach and later discusses how this toolset can be used to acquire a CIM for a business system. To enable the IDM approach the author has implemented a prototype of the IDM toolset, which will be discussed later in this section.

3.1. Scope of the IDM Toolset

This sub-section defines the scope of the IDM toolset and discusses the involved tools, artifacts and their purpose in context of MDA life cycle. Also author outlines who are the users of the toolset and what kind of roles are involved in the process of developing a domain model.

In earlier authors work [79] and [83] some suggestions have been made on what tool support would be necessary for the construction of the TFM. Since one of the goals of this work is to acquire a TFM, it is clear that first of all a tool to support a TFM is necessary to enable the users to model the TFM. This is not the only tool necessary to support the IDM approach. Once there are means to construct a TFM, further tool support would need to provide the possibility to assist the user with the task of acquiring a CIM. Considering the IDM approach first described in authors paper [85] and in more detail in previous section “The Integrated Domain Modeling Approach”, the scope of the required toolset can be further discussed. This approach suggests starting the system analysis process from formally defined declarative and procedural knowledge with a perspective of integration with MDA. IDM is exploiting

Ontology and Use Cases for defining the knowledge model for a business domain. A knowledge engineer constructs the Ontology and a system analyst together with the business constructs Use Cases. While doing so the Use Cases need to be validated in order to correspond to the Ontology. This is an iterative process, because the Ontology or the Use Cases have to be modified until they correspond to each other. This process requires a sufficient supporting toolset, so that the correspondence can be automatically determined sequentially in each step of the knowledge model development.

The IDM toolset needs to support the user to do the following main activities:

1) Construct or reuse an existing domain Ontology;
2) Develop Use Cases model describing the business processes of the domain;
3) Validate the Use Cases model by using natural language processing and the domain Ontology;
4) Automatically generate the CIM for this domain in form of a TFM;
5) Allow the user to further model the TFM of the domain by adding main functional cycle and logical relationships;
6) Validate each of the models against the corresponding meta-model.

The users of this toolset would be the knowledge engineer and/or the system analyst who would work together with the business to gather and validate the input knowledge and resulting domain model. The knowledge engineer and the system analyst may or may not be the same person. The main tasks of the knowledge engineer are to build the Ontology corresponding to the business domain, validate the Use Cases to correspond to the Ontology, and in case they do not – correct the Ontology or suggest changes in the Use Cases. With Ontology it is possible to define the declarative knowledge or in other words the dictionary of the domain. This is where the concepts, terms and their meaning are defined. This is done based on existing business documentation and in close contact with the business team to clarify and validate the Ontology. In IDM Toolset context, any tool that supports OWL standard for Ontology can be used, for example, Protégé. Later the Use Cases Editor tool will use the acquired OWL artifact.

On the other hand, the main tasks of system analyst are to build the Use Cases corresponding to the business domain, validate Use Cases to correspond to the

Ontology, and in case they do not – correct the Use Cases or suggest changes to Ontology, transform the Use Cases to the TFM and together with the business team validate if they correspond to the business domain. Procedural knowledge is recorded in the form of Use Cases showing the scenarios with steps, alternative scenarios and conditions for some business domain objectives. Again working closely with the business team the user needs to record the Use Cases using the Use Cases Editor. When the Use Cases artifact is acquired it can be validated against the Ontology to check for unambiguity and consistency. For unambiguity the terms and concepts are compared with the nouns used in the Use Cases. And comparing Ontology properties and noun/verb combinations checks consistency. Next step in Use Cases validation is to check if it corresponds to the tools meta-model.

Validation of the Ontology and Use Cases have to take place from both perspectives – to check if Ontology is not missing anything and also to check if Use Cases correspond to the Ontology. When validating the TFM new changes might come to all models – Ontology, Use Cases and TFM, since the TFM is the graphical representation of the business domain and the business team can analyze more effectively. During each stage of CIM construction cross artifact validation should take place. For example, while creating a step in the Use Cases, if the user finds out that a term is missing or is incorrect in the Ontology he should do modifications. Another example is related to cause/effect relationships. These are more apparent in the TFM; thus after acquiring the TFM the user might find that some topological relationships are incorrect, so he should return to the Use Cases and do the corresponding modifications. This is an iterative process until Use Cases correspond to the Ontology and the TFM corresponds to the Use Cases. The TFM Editor also allows the user to identify the main functioning cycle, sub-cycles and add logical operations, which are not part of the model transformation. These elements add more details to the CIM and allows to validating the models again from a different perspective – process topology and cycles. This is the CIM level within MDA, which represents the AS-IS state of a business system. At different stages of the CIM development the business team should be consulted to validate the models and finally the CIM should be signed-off by the business executive. When the sign-off is done, the CIM is approved and the transformation to PIM/PSM can begin.

As shown in Figure 3.1 the toolset provides the artifacts necessary for input of further model transformation to UML. Thus providing a head start for the technical documentation of software solution. The technical team would be responsible to execute the transformation and validate results. The author does not provide the further transformation path in detail since it is out of scope of this research. In short, the technical team would work on the PIM/PSM level, generate source code and continue with the implementation of the software solution. In case of any inconsistencies with CIM are discovered during the PIM/PSM modeling or implementation the technical team needs to give feedback to the system analyst and knowledge engineers to make changes on the CIM level. Then the transformation from CIM to PIM/PSM needs to be executed again so that the models are consistent. After the changes have been generated to the PIM/PSM and source code, the technical team can continue with the implementation.

There are various MDA tools available in the area of PIM/PSM. It is possible to start with the UML diagrams (PIM level), then add some platform specific features with OCL (PSM level), and then based on this model it is possible to generate the source code. In the context of TFM in the PIM/PSM level UML is used. In related research there is specific Topological UML (TopUML) profile developed. The process of transforming TFM to various TopUML models is described in related research [12]. Having a supporting tool for a UML profile is common practice and MDA frameworks fully support this approach. Usually for a particular UML profile there is a corresponding tool for constructing models, which correspond to the UML profile. In case of TopUML there also has to be such a tool, but on top of that, there will also be a set of model transformations from TFM to TopUML models.

Although model transformation and source code generation is considered to be one of the main benefits of MDA, one should not underestimate the modeling process itself even if there is no tool support at all and no automatic generation of source code. By following the transformation path and validating the artifacts at each step, the process helps to ensure consistency with the business domain. Also at this point, it is also possible to prepare documentation based on the TFM, Use Cases and Ontology for the executive to approve the domain model by sign-off or give it back for another iteration to the knowledge engineer, system analyst and the business team. There can

be different types of documentation, which include the analysis of the business domain for different types of software engineering approaches. Nevertheless, some kind of documentation will be necessary in most cases to serve as a formal contract between the customer and supplier, or as a general domain documentation for business partners and internally.

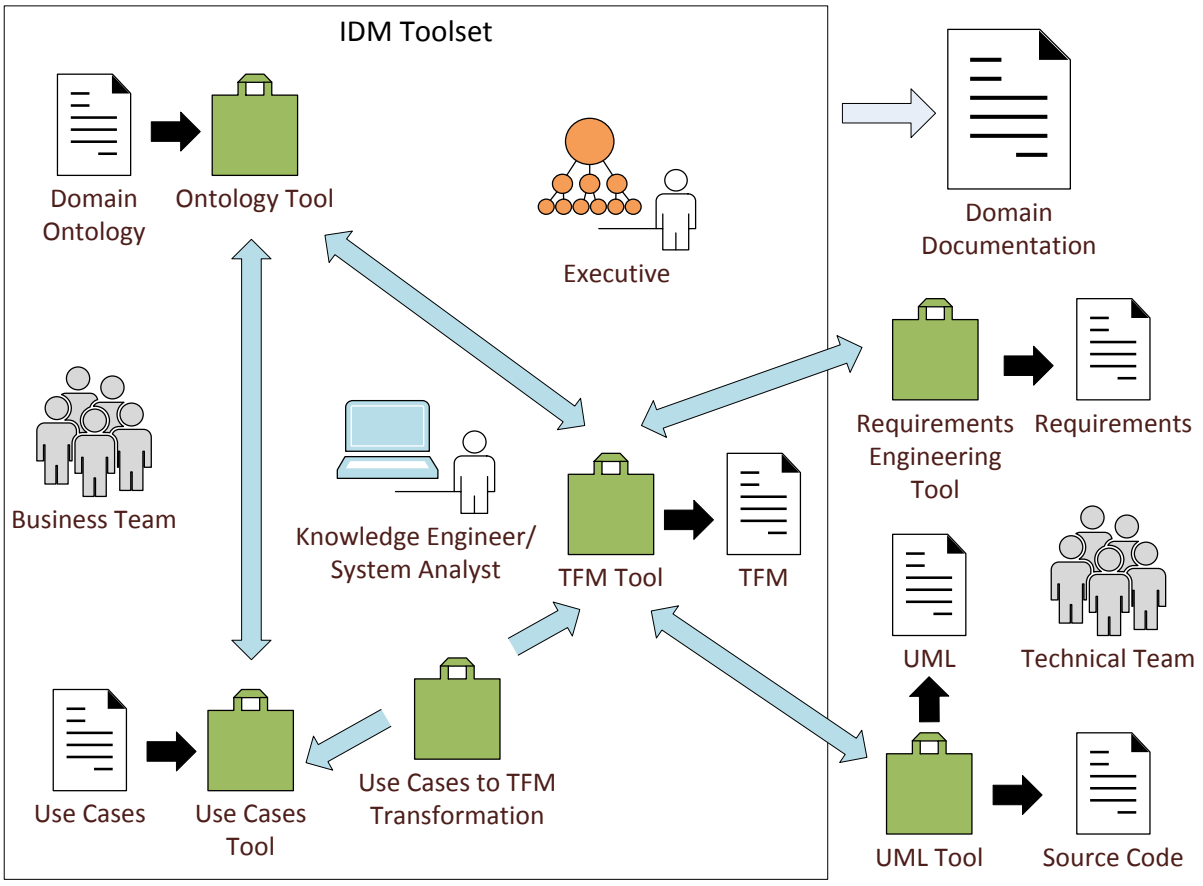


Figure 3.1. Scope of the IDM toolset

In Figure 3.1 the author shows tools, artifacts and roles involved within the scope of the IDM Toolset and beyond. Document icon represents the artifacts – domain Ontology, Use Cases, TFM, UML, source code, requirements and domain documentation. Briefcase icon represents the tools – Ontology tool, Use Cases tool, Use Cases to TFM transformation, TFM tool, requirements engineering tool, and UML tool. Black arrows show the artifacts of the tools. Blue arrows represent the interaction between tools. The roles involved are represented with person icons – knowledge engineer, system analyst, business team, executive and technical team. Grey arrow shows the relationship between IDM Toolset and documentation. The IDM toolset consists of: 1) Ontology tool; 2) Use Cases tool; 3) TFM tool; 4) Use Cases to TFM

transformation. Author gives a description of each of the tools in the following sections and then goes through the details of developing the corresponding tools. Details on the requirements engineering and UML tools are given in the "Out of Scope" section.

3.1.1. Ontology Tool

Ontology tool is meant for defining Ontology according to OWL standard. It can be any tool that supports the OWL standard, for example, Protégé (an Eclipse based Ontology tool) [75]. This tool in the context of IDM approach is going to be used for the following:

1) Create class hierarchy of an Ontology;
2) Create properties of an Ontology;
3) Import Ontology in OWL format into IDM toolset.

3.1.2. Use Cases Tool

Use cases tool should allow the user to define the Use Cases for a business domain and check if they correspond to the Ontology. There is no such tool available, thus the author has developed an Eclipse based tool called Use Cases Editor. This tool needs to have the following functionality:

1) Support the Use Cases model format corresponding to the Use Cases meta-model;
2) Allow to model the Use Cases, i.e., create Use Case elements and define their properties;
3) Validate the model according to the Use Cases meta-model;
4) Represent the model in a graphical way instead of plain XMI;
5) Upload an Ontology in a form of OWL and validate the Use Cases model against it;

Although the methodology for validating Use Cases against Ontology is provided in section 2, this functionality (number 5 in the list for Use Cases Tool) will

not be implemented as part of this thesis for time constraint reasons. This step will have to be done manually by hand, without tool support.

3.1.3. TFM Tool

TFM tool's purpose is to allow the user to model the CIM of a business domain manually creating each element, but most importantly to provide means to support a TFM format corresponding to XMI. There is no such tool available, so the author has developed two Eclipse based tools called TFM Editor and TFM Diagram. In more detail the TFM Editor tool needs to have the following functionality:

1) Support the TFM model format corresponding to the TFM meta-model;
2) Allow to model the TFM, i.e., create TFM elements and define their properties;
3) Validate the model according to the TFM meta-model;
4) Represent the model in a graphical way instead of plain XMI.

And the TFM Diagram tool in addition needs to display the TFM in a form of graph diagram and provide a palette for the user to model TFM in diagram form.

3.1.4. Use Cases to TFM Transformation

The purpose of Use Cases to TFM transformation is to enable a model transformation between Use Cases and TFM. This is done by fetching the functional features and topological relationships from Use Cases, and then generating the TFM for a business domain. There is no such tool available, so the author has developed an Eclipse based tool called UseCases2TFM Transformation. This tool needs to have the following functionality:

1) Support the Use Cases model as source and TFM model as target model according to their meta-models for the model transformation;
2) Fetch functional features, topological relationships and actors from the Use Cases model;

3) Analyze Use Cases steps with natural language processing (NLP) to identify the entity and the description of the functional feature;
4) Integrate seamlessly with Use Cases Editor and TFM Editor.

3.1.5. Out of Scope

There are 2 tools seen in Figure 2.1 that are out of the IDM toolset scope – the requirements engineering tool and the UML tool. The goal of this research and the IDM approach is to provide means for CIM construction in a formal way based on the available standards and software engineering practices. The boundaries of the scope for IDM toolset are based on the boundaries of the CIM within MDA. Thus the IDM toolset's model transformation output and the endpoint is the TFM. This is the scope considered in this research so far and is used for this thesis, but the author might expand the scope in future research. The future work could include transformation to PIM/PSM from TFM also within the IDM toolset, and importing software requirements from a requirements engineering tool for business process validation and software scope definition. Also as mentioned in section 3.1.2, the functionality to validate the Use Cases model against the Ontology will not be implemented as part of the Use Cases Tool. IDM Toolset is a high effort implementation project with many components. For time and effort limitations of this research, the author is leaving this for future work. The validation methodology is provided in section 2 and will have to be done manually by the user without an additional tool support.

After acquiring a CIM the next step in MDA lifecycle is transforming CIM to PIM/PSM. The source for this transformation is the TFM and the target is UML. All UML diagrams could be considered, but the main focus should be on the transformation to UML class diagram. This is because UML class diagram is the basis of the object oriented software architecture and MDA tools are available to generate the corresponding source code. However, if any intermediate diagrams are necessary to achieve a model transformation from TFM to UML class diagram, then intermediate transformation should also be implemented sequentially and used to assist the transformation to the class diagram. In his thesis on Topological UML Uldis Doniņš has done research touching these topics [13]. He outlined possible model

transformations from TFM to TopUML and described steps that have to be done manually. However, implementing a formal model transformation with a model transformation language and the tool support is an open question.

For this transformation to take place first of all a UML tool is necessary to serve as the facility of the target model. Here there are several options to choose from existing open source UML tools based on Eclipse modeling frameworks, like, Eclipse Papyrus [15]. Thus it is also possible to reuse the available UML meta-model. For the source model the existing IDM toolset's TFM Editor and Diagram Tool can be used. But the most important step is to define a formal model transformation from TFM to UML class diagram. The level of automation may vary, however here is a possible logic for the transformation:

1. Manually add method names to the functional features in the TFM with TFM Editor/Diagram Tool (additionally it would also be possible to specify the parameters of the method);
2. Trigger the model transformation to UML class diagram:
    a. Loop over functional features and create classes defined in the "object" attribute with the corresponding methods defined in the "actions" attribute (for this there might be a new attribute introduced) adding the method to the corresponding class;
    b. Check if getters and/or setters (get and set methods from object oriented programming) are present as methods in the classes:
        i. For each getter there should be a setter and vice versa;
        ii. If there is a setter of a particular attribute of type defined as a class, then a relationship should be created between the classes.
        iii. If there is no class for the attributes type then create an attribute.
3. After model transformation has been executed there are some steps the user has to do manually to finalize the UML class diagram:
    a. Check for missing attributes and set the types;
    b. Check for missing methods and set the return type;
    c. Check for missing classes and relationships;

d. Set the inheritance (inheritance could also be picked automatically from the Ontology used in the IDM toolset).

The other future task is to enable support for requirements engineering tools for the TFM Editor. Research on TFM4MDA [62] talks about mapping the functional requirements to the functional features in order to define the scope of the software as well as to validate the business process and TFM. After acquiring the requirements for a software is necessary to do such mapping to ensure that the business processes support the provided requirements. There might be a mistake in the requirements done by the system analyst or the business team failed to provide the right input. Also it is not a rare case when business processes need to be changed and/or new processes implemented after deploying new software solutions. By doing the mapping of functional requirements to functional features of the TFM any deviations can be formally identified and acted upon. There are many requirements formats used in requirements engineering, but the author is particularly interested in the Requirements Interchange Format (ReqIF) format defined by the Object Management Group [51]. ReqIF defines an open, non-proprietary exchange format for requirements, which meant to provide a common format for requirements. For example, the collaboration between partner companies can be improved by the benefits of applying requirements management methods across company borders even if the partners use different requirement engineering formats and tools. Nonetheless, this common format can also be used directly when building the software requirements. This can be done with Eclipse ProR tool based on the Eclipse Requirements Modeling Framework (RMF) [16], which is an implementation of the ReqIF. The IDM toolset should make use of ReqIF for requirements mapping. Author stresses again the importance of using standards and available modeling frameworks because this kind of opportunity would not be provided if a completely custom tool would be built from scratch.

This is a rough draft of the possible future work and needs more effort to be designed and implemented in real life. However, it would not be possible to think about these possibilities without the IDM approach and the supporting

toolset. IDM toolset provides the infrastructure for further research and developments.

3.2. Architecture of the IDM Toolset

This sub-section introduces the architecture of the IDM toolset in detail. The IDM approach deals with the knowledge gathering and analysis tasks at the beginning of MDA life cycle: 1) defining a formal data structure for the knowledge about the system; 2) retrieving functional features for TFM; 3) retrieving topology or cause-effect relations between functional features, thus constructing a TFM for the system.

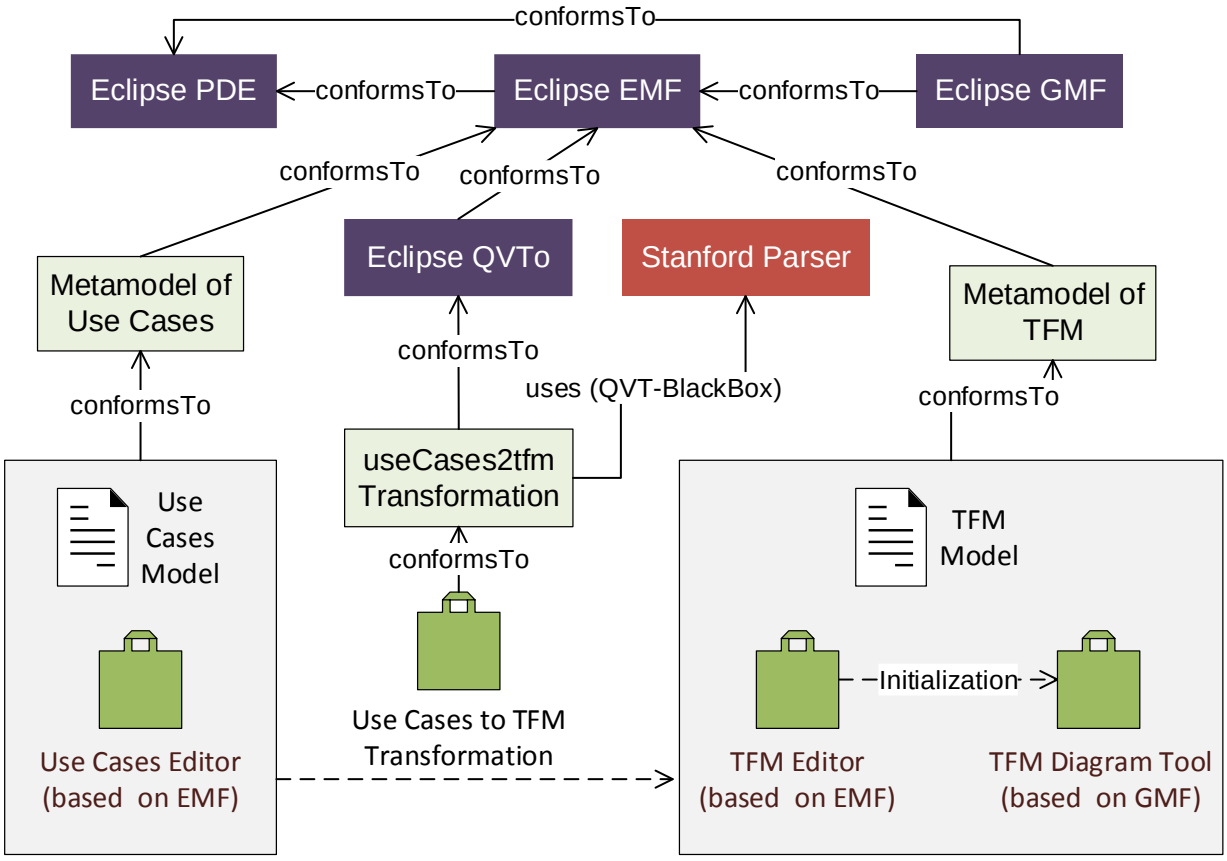


Figure 3.2. Architecture of the IDM toolset

As discussed earlier, the IDM approach exploits Use Cases with a specific template to define the knowledge about the system. This is a promising solution because Use Case driven development is a widespread approach in software engineering. Another advantage is that a MOF compatible meta-model can be created for this Use Case template using XMI. Author has published the Use Cases meta-model in [84] and the meta-model for TFM published in [81]. A statistical parser can

be used for analyzing the sentences of Use Cases, and thus retrieving data for functional features to construct the TFM of the business system. For retrieving the cause-effect relations between these functional features the structure of the Use Cases can be used. Thus there are 2 meta-models – a meta-model for Use Cases (a set of Use Cases) and a meta-model for TFM. Then there is a method to transform Use Cases into a TFM. The method can be expressed as an exogenous model transformation [42] because the source and target models have different meta-models. At this point OMG's standard for model transformations QVT comes into the picture. QVT provides the necessary domain-specific languages to define the transformation between Use Cases and TFM, including a QVT/BlackBox mechanism for integrating existing non-QVT libraries or transformations like a statistical parser. There are some workable implementations of QVT like Eclipse M2M's QVTo model transformation language.

In Figure 3.2 author shows the architecture of the IDM toolset. Document icon represents the artifacts – Use Cases model and TFM model. Briefcase icon represents the tools – Use Cases Editor, Use Cases to TFM transformation, TFM Editor, and TFM Diagram Tool. The arrows show conformity to the meta-models, Eclipse frameworks or use of a 3rd party Java library. The intermittent arrows show the transformation direction and also the initialization of the diagram. This excludes the 3rd party OWL tool, for which the Protégé tool is used. The following Eclipse frameworks are used – PDE, EMF, GMF and QVTo. The conformity is also shown. Since EMF and GFM allows generating Eclipse plug-ins it conforms to the PDE, which can be later used to extend the functionality of the tools. GMF and QVTo (QVT Operational) both of course conform to the EMF, which they are based on. In addition, a 3rd party statistical parser (Java library) is used for natural language processing – Stanford Parser. The following tools were developed by the author – Use Cases Editor, TFM Editor, TFM Diagram Tool and Use Cases to TFM Transformation. Here the author shows the technical tools, which user might not even be aware of while using the toolset. For example, the model transformation is happening in the background after a left mouse click on the Use Cases model and transformation execution. The definition of the model transformation itself is part of the Use Cases to TFM Transformation and conforms to QVTo. The Use Case Editor initially is generated by

the EMF [17] based on the meta-model of Use Cases, which is published by the author in [84]. Later some modifications are done in the generated code, e.g., to change the default representation of the model in the model editor. These changes are marked with a special annotation ("@Generated NOT") so that possible changes in the meta-model in the future could still be generated and the custom code would not be overwritten.

3.3. Use Cases Editor

This sub-section talks about the Use Cases Editor implementation as an Eclipse plug-in done by the author, which is based on the Eclipse Modeling Framework (EMF). Further the author guides through the development steps and provides screenshots of the configuration and source code snippets. The full source code and configuration can be found in the appendix.

3.3.1. Use Cases Meta-model

The first and the most important step is to develop a Use Cases Editor is to define the Use Cases meta-model. A Use Case is a description of a process (as defined in Section 1.2.5) and its steps in detail, and may be worded in terms of a formal model. A Use Case is intended to provide sufficient detail for it to be understood on its own. A Use Case has been described as "a generalized description of a set of interactions between the system and one or more actors, where an actor is either a user or another system [20]. There is no standard way to write the content of a Use Case, and different formats work well in different cases [22]. But there is a common style to use: 1) Title: "goal the Use Case is trying to satisfy"; 2) Main Success Scenario: numbered list of steps; 3) Step: "a simple statement of the interaction between the actor and a system"; 4) Extensions: separately numbered lists, one per Extension; 5) Extension: "a condition that results in different interactions from the main success scenario". In the Unified Modeling Language (UML), the relationships between all (or a set of) the Use Cases and actors are represented in a Use Case diagram or diagrams, originally based upon Ivar Jacobson's Objectory notation [22].

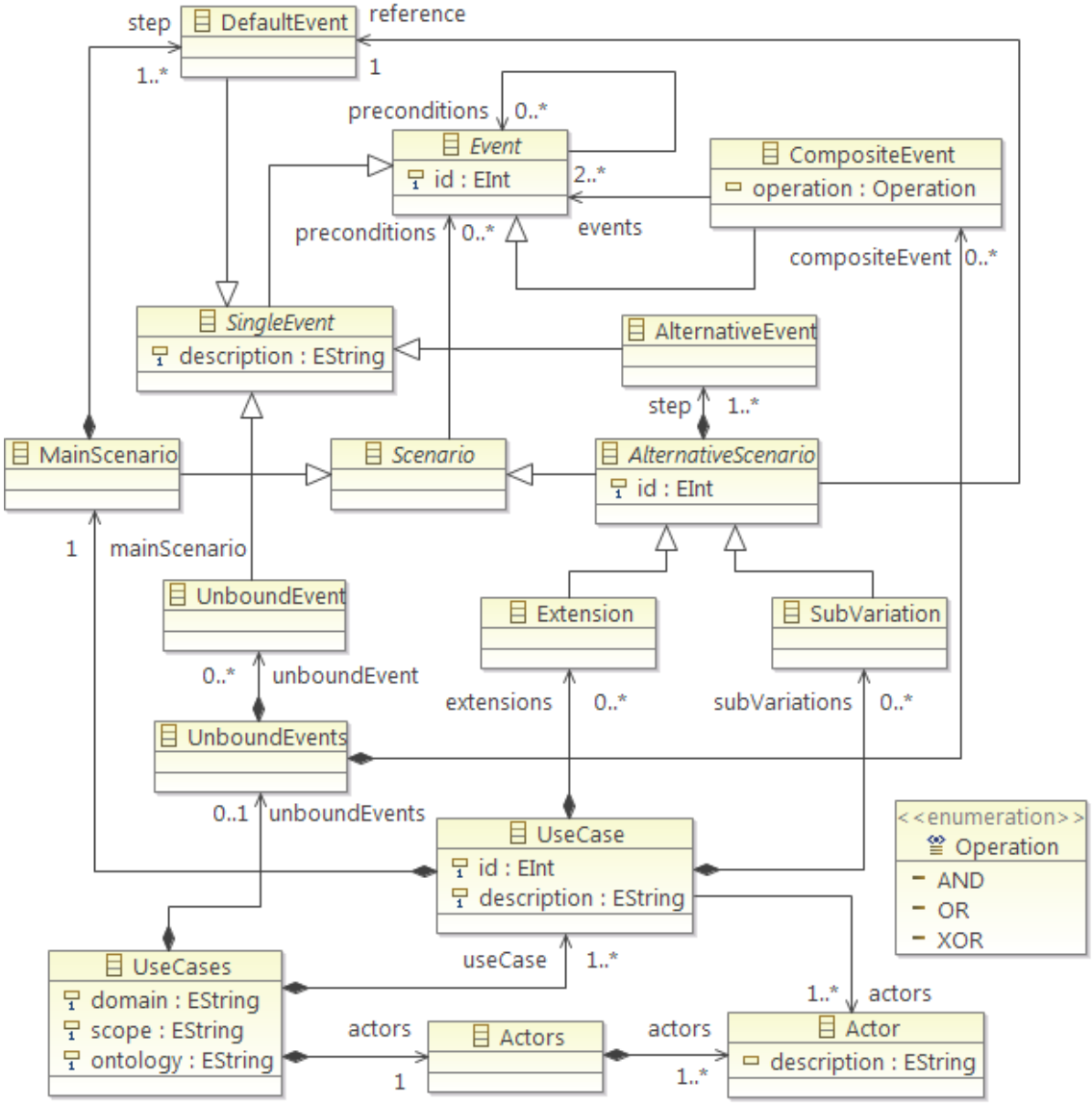


Figure 3.3. Use Cases Meta-model

Figure 3.3 shows a MOF-compatible meta-model for a set of Use Cases developed by the author. A set of Use Cases consists of 1 or more Use Cases, which have a main scenario, extensions and sub-variations, which consist of Use Case steps. Each Use Case has a title, list of actors (at least 1 actor), and can have preconditions. Each Use Case step has its number description and can have a precondition. For extensions and sub-variations the reference attribute is used. This shows which of the steps in the main scenario it references, so for main scenario steps reference will be empty. Full XMI of this model can be found in the Appendix 3 giving all details on the configuration.

For the IDM Use Cases Editor the author has developed a Use Cases meta-model shown in Figure 3.3. Main classes for the Use Cases model are Actor, Event, Scenario and UseCase, which tie the previous objects together. Use cases class is the root class for the model and has attributes domain, scope and Ontology. Ontology attribute will hold the technical name of the Ontology for the domain, which will be uploaded via Use Cases tool. As shown in the meta-model, Use Cases model consist of actors, unbound events and Use Cases.

Actors are included in this model to organize the actors involved in the Use Cases, so that it would be possible to choose from already existing actors or add new ones. Class Actors is a container for actors in the Use Cases model, so both actor's references are containment. Actor has a description, describing what this actor represents in this domain. There is only 1 container in a Use Case model, but there can be many actors. There has to be at least 1 actor in the Use Cases model.

Events are all the steps (this terminology is used because each step results in an event that a particular step has been executed, so for the purpose of this toolset this definition fits) in the Use Case and also all preconditions. Event is an abstract class with an id as attributes. Attribute id has to be unique in scope of all events, so that it is possible to unambiguously reference an event. Each event can have 0 or many preconditions, which are also events. Class Event has 2 sub-classes: SingleEvent and Composite Event. SingleEvent is an abstract class representing a single event. It has a description attribute and 3 sub-classes: DefaultEvent, AlternativeEvent and UnboundEvent. Class DefaultEvent represents an event that occurs in the default sequence of events of a Use Case. Class AlternativeEvent represents an event that occurs in an alternative sequence of events of a Use Case. So the main scenario uses the default events as steps and extensions and sub-variation use the alternative events as steps. Class UnboundEvent represents an unbound event that is not used in any scenario, but is used as a precondition. All preconditions for events or scenario steps also have to be events, but some events will not be part of a scenario in the Use Cases model. For this kind of events we have the unbound event container. Class UnboundEvents is a container for unbound events in the Use Cases model.

Composite events are represented with the class CompositeEvent. This kind of event let's you reference other events (at least 2) with a corresponding operation.

Operations are defined in an enumeration Operation, which defines AND, OR and XOR operations. This way it is possible to define sequences of events with operations. Moreover, these sequences can also contain other composite events. The unbound events container will also hold composite events.

Scenarios hold the preconditions and the order of the events that occur if these preconditions are true. Class Scenario is an abstract class, which has 2 sub-classes MainScenario and AlternativeScenario. MainScenario is a container for the default sequence of events happening for a particular Use Case, therefore main scenario can have 1 or more default event objects. There can be only 1 main scenario for a single Use Case. AlternativeScenario is an abstract class, which represents a possible alternative sequence of events (alternative to the main scenario). Alternative scenarios have an id attribute, because there can be 0 or more than 1 alternative scenarios in a single Use Case. There are 2 possible alternative scenarios – extension or sub-variation of the main scenario.

Use case holds the references to the scenarios and actors. There has to be at least 1 Use Case in a Use Cases model. Class UseCase also has an id and a description as attributes. Each Use Case must have 1 main scenario and it may or may not have extensions or sub-variations.

3.3.2. Generating Source Code with EMF

The next step is to generate the source code of the Use Cases Editor with Eclipse Modeling Framework. EMF requires the meta-model to be created in the form of Ecore model. Since the author was using Ecore Tools to develop the Use Cases meta-model in the first place, the corresponding Ecore model was automatically composed. In Figure 3.3 the corresponding ".ecorediag" diagram format is shown and the ".ecore" model is shown in Figure 3.4. Ecore is raw format of the meta-model used by the EMF.

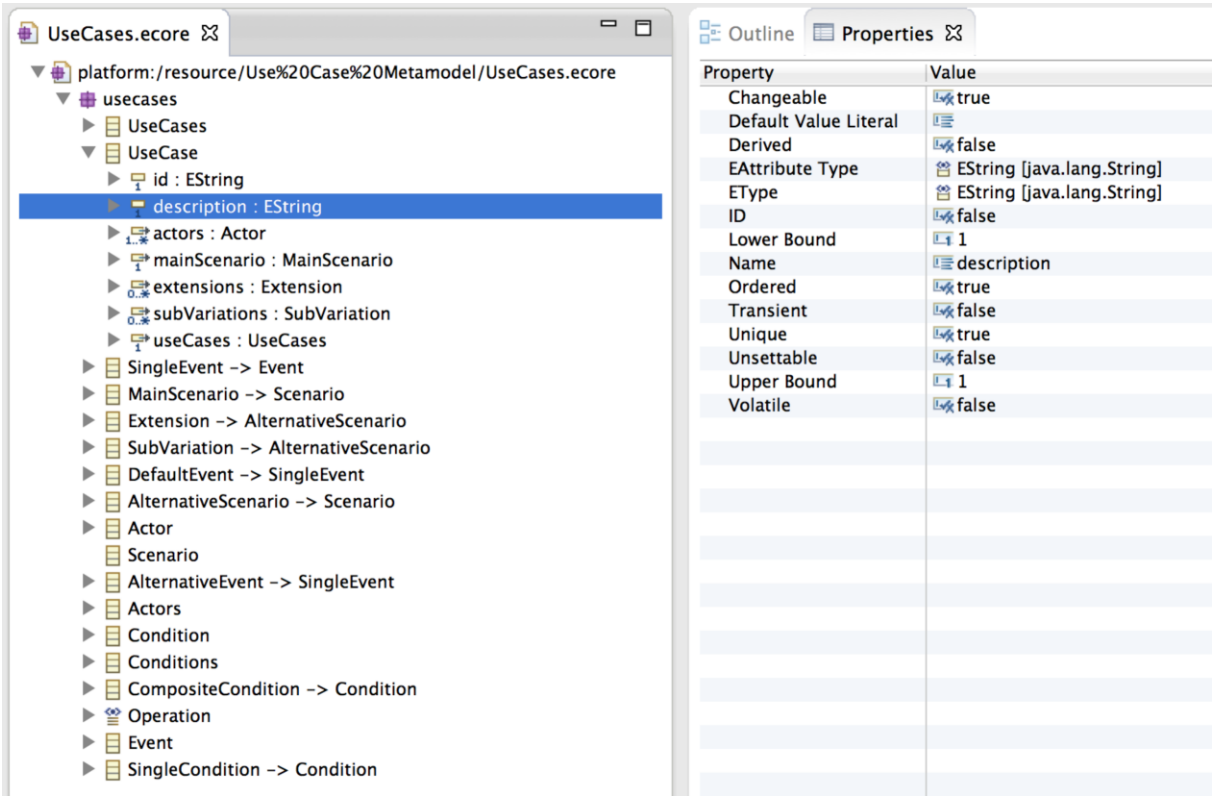


Figure 3.4. Use Cases Ecore Model

Figure 3.4 shows a MOF-compatible meta-model in raw form of Ecore for a set of Use Cases. This Ecore model exactly corresponds to the Ecore diagram in Figure 3.3. Full XMI of this model can be found in the Appendix 3 giving all details on the configuration. Once there is a meta-model defined in the form of Ecore, the next step is to develop a generation model for the EMF generator. The initial Generation Model (EMF special model) can be generated automatically, which is responsible for the configuration of model editor tool that will be generated corresponding to the meta-model. The resulting model editor consists of 4 Eclipse plug-ins built according to Eclipse PDE – Model, Edit, Editor and Test. With the generation model it is possible to set some attributes for the source code generation of these plug-ins, but for the IDM Toolset the author uses the default values. For example, it is possible to set the plug-in class and source directory.

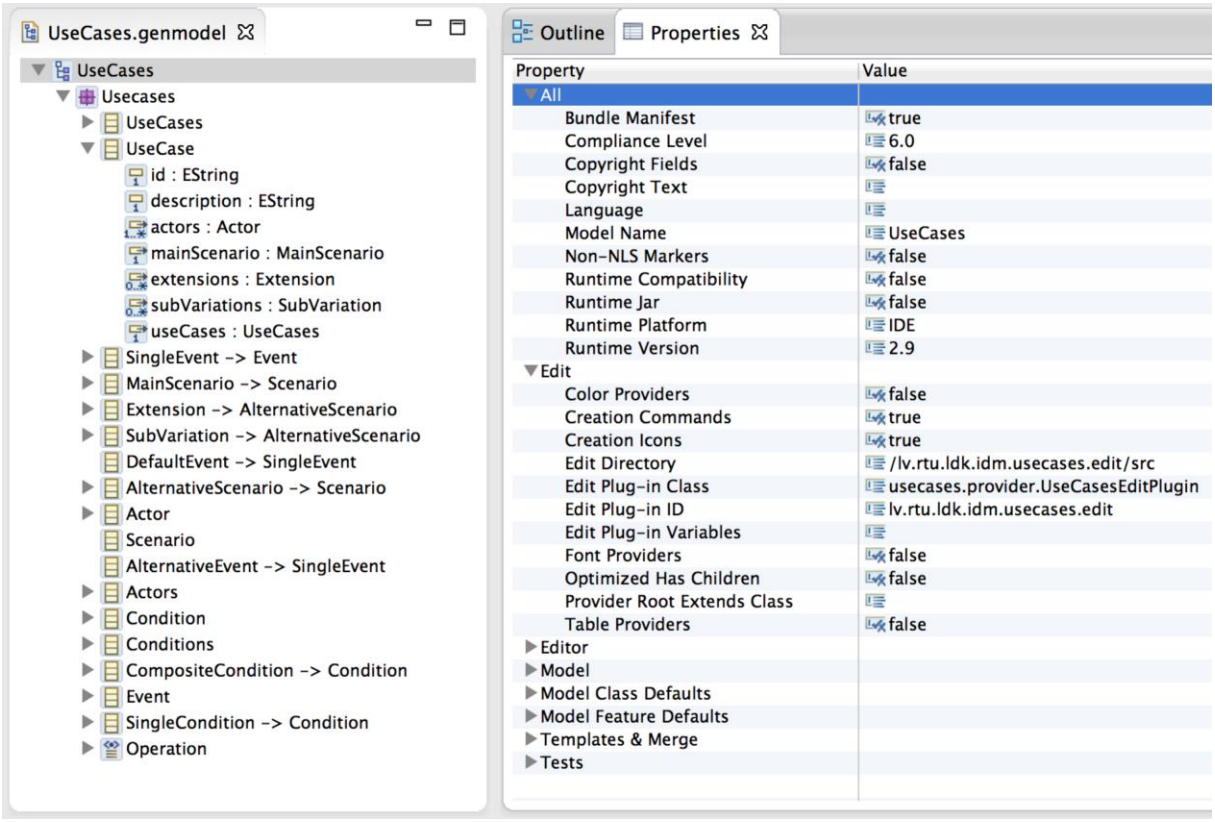


Figure 3.5. Generation model for Use Cases Editor

Figure 3.5 shows the generation model of EMF for Use Cases Editor. The root node of the model represents the model editor and the child node contains the meta-model element definitions for the tool. Full XMI of this model can be found in the Appendix 5 giving all details on the configuration.

3.3.3. Implementation

Using this generation model the initial source code for the Use Cases Editor has been generated. The classes and interfaces generated by EMF are marked with a special Java annotation "@generated". It is possible to make changes to this initial source code by changing the annotation to "@generated NOT". Thus the author can add his own implementation to some of the generated classes. For the particular Use Cases Editor this was done several times in order to change the default text output of the model elements. For this it is necessary to modify the provider classes for the particular element, which can be seen in Figure 3.6. For example, for the element

“Composite Condition” there is a corresponding provider class CompositeConditionItemProivder.java.

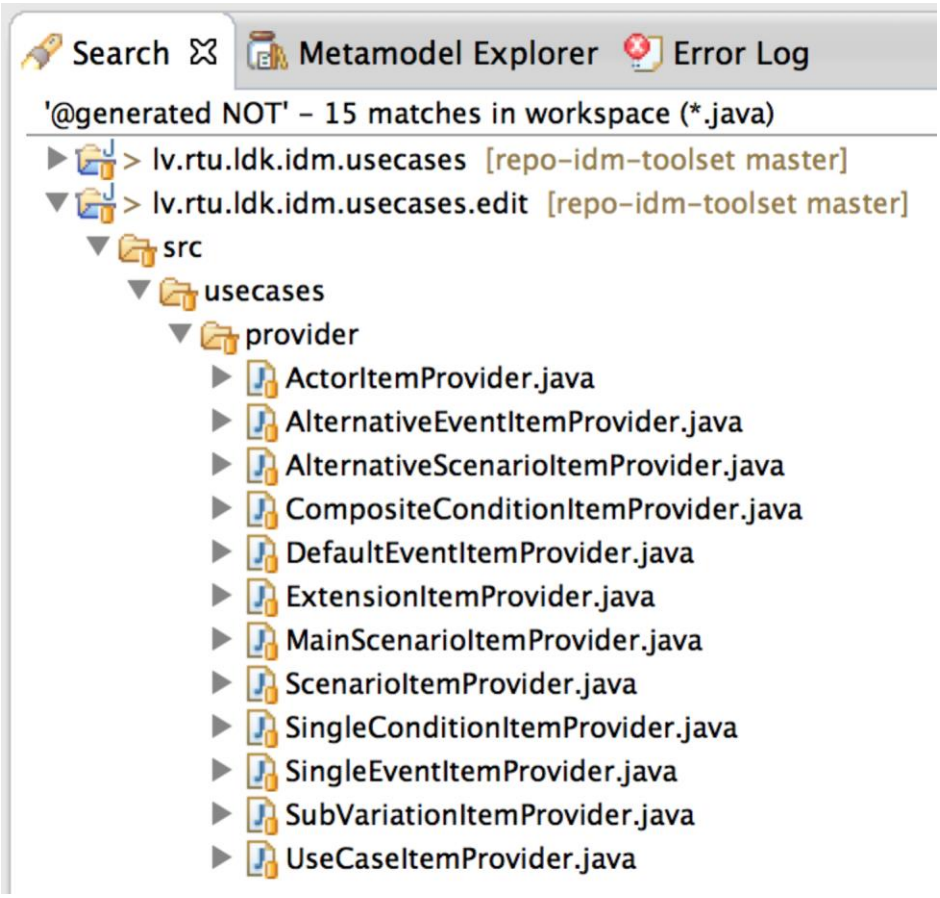


Figure 3.6. Use Cases Editor Custom Classes.

Figure 3.6 shows the modified classes of the generated Use Cases Editor, where the author changed the default implementation. The class CompositeConditionItemProivder.java has a method to return the text of the element – getText. This method receives an object of type CompositeCondition. The goal is to return each of the single condition texts and the operations between. In case of other composite conditions the method puts the text “(…)”. As it is shown in Figure 3.6 the method iterates over conditions got from the composite condition, and composes the resulting string.

CompositeConditionItemProvider.java

```
124     /**
125      * This returns the label text for the adapted class.
126      * <!-- begin-user-doc -->
127      * <!-- end-user-doc -->
128      * @generated NOT
129      */
130     @Override
131     public String getText(Object object) {
132
133         CompositeCondition compositeCondition = (CompositeCondition) object;
134         EList<Condition> cEvents = compositeCondition.getConditions();
135         String preconditionString = "";
136
137         Iterator<Condition> cIt = cEvents.iterator();
138         while (cIt.hasNext()) {
139             Object cEvent = cIt.next();
140             if (cEvent instanceof SingleCondition) {
141                 preconditionString = preconditionString
142                         + ((SingleCondition) cEvent).getDescription();
143             } else if (cEvent instanceof CompositeCondition) {
144                 preconditionString = preconditionString + "(...)";
145             }
146             if (cIt.hasNext()) {
147                 preconditionString = preconditionString + " "
148                         + compositeCondition.getOperation().getName() + " ";
149             }
150         }
151
152         return preconditionString + " (" + compositeCondition.getId() + ")";
153     }
```

Figure 3.7. Use Cases Editor Custom Source Code.

Figure 3.7 shows the getText method of CompositeConditionItemProvider.java class. The method is marked with the "@generated NOT" annotation for the EMF generator. This method receives an object, iterates over conditions and returns a string to be shown in the Use Cases Editor for a composite condition. Similarly also the other item provider classes are modified to the needs of the Use Cases Editor tool. Another thing that author changes for the purpose of IDM toolset is the default icon set provided by the EMF. For each of the elements, which are visible in the Use Cases Editor a special icon is provided to represent the element and help the user to understand the meaning of each element. Freely available icon set was reused and slightly modified for IDM toolset. It is possible to see the icons in Figure 3.8.

3.3.4. Resulting Use Cases Editor

After source code generation it is possible to run the Use Cases Editor tool and try it out. It is dependent on these Eclipse plugins – Use Cases ("lv.rtu.ldk.idm.usecases"), Use Cases edit ("lv.rtu.ldk.idm.usecases.edit"), Use Cases Editor ("lv.rtu.ldk.idm.usecases.editor") and Use Cases Test ("lv.rtu.ldk.idm.usecases.tests").

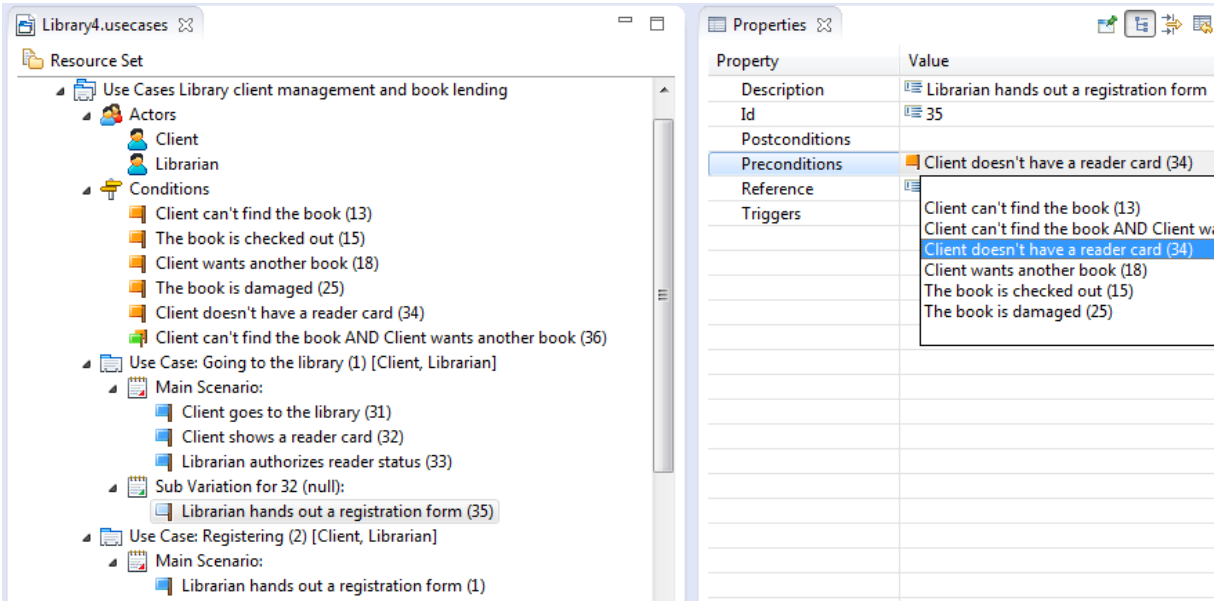


Figure 3.8. Use Cases Editor tool

Figure 3.8 shows the Use Cases Editor tool. On the left side there is a Use Cases model constructed by the user and on the right the properties view of a Use Cases step, where it is possible to set preconditions. There are specific icons for each of the elements of the meta-model. This tool allows the users to define Actors, Conditions, Composite Condition, Use Cases, Main Scenarios, Alternative Scenarios and their steps. To create an element the user needs to right click on the Use Cases root node and create a child element of the desired type. For some of the element types a container node is necessary – actors, conditions, Use Case. Inside actors container it is possible to create actors elements. Inside conditions container it is possible to create conditions and composite conditions. At least 2 regular conditions have to be defined before, so that they can be referenced by a composite condition; thus a composite condition can be created with a logical operator. Underneath a Use Case node can

contain the following container nodes – main scenario, sub variation, extension. At least the main scenario has to be defined for a Use Case. Inside the main scenario container the user can create default event elements. And inside the sub variation or extension containers the user can create alternative event elements. To set the attributes of an element the user must use the properties pane. The file extension of the Use Cases artifact is ".usecases". This file can also be opened with a text editor and then it is possible to see that underneath it has an XMI format.

3.4. TFM Editor

The Use Cases Editor tool is meant for constructing the Use Cases model. Similarly, the TFM Editor tool is meant for constructing the TFM model. This sub-section talks about the TFM Editor implementation as an Eclipse plug-in developed by the author. Author provides screenshots of the development snippets and configuration. The full source code and configuration can be found in the appendix.

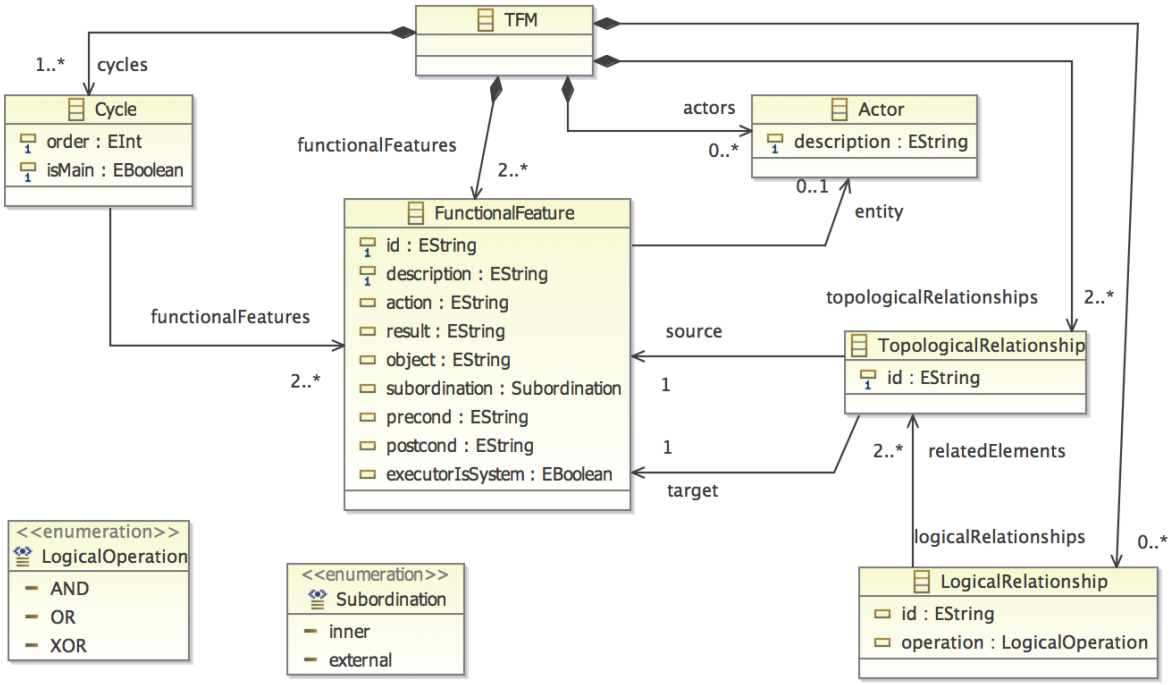


Figure 3.9. TFM Meta-model.

The steps to develop the TFM Editor are similar to the Use Cases Editor since it is also an EMF based model editor. The first step is the TFM meta-model. TFM Editor

is based on EMF and conforms to the latest TFM meta-model published for related TFM research in doctoral thesis of U. Donins [13]. Author developed a meta-model for TFM in EMF Ecore format. Figure 3.9 shows a MOF-compatible meta-model for a TFM developed by the author. The main classes of the TFM meta-model are TFM, Cycle, Functional Feature, Actor, Topological Relationship and Logical Relationship. The meta-model also has 2 enumerations – Logical Operation and Subordination. Full XMI of this model can be found in the appendix giving all details on the configuration. TFM class is the root class for the model. It is a container for the rest of the objects. TFM can consist of 1 or more cycles, at least 2 functional features, at least 2 topological relationships, and can contain actors and logical relationships.

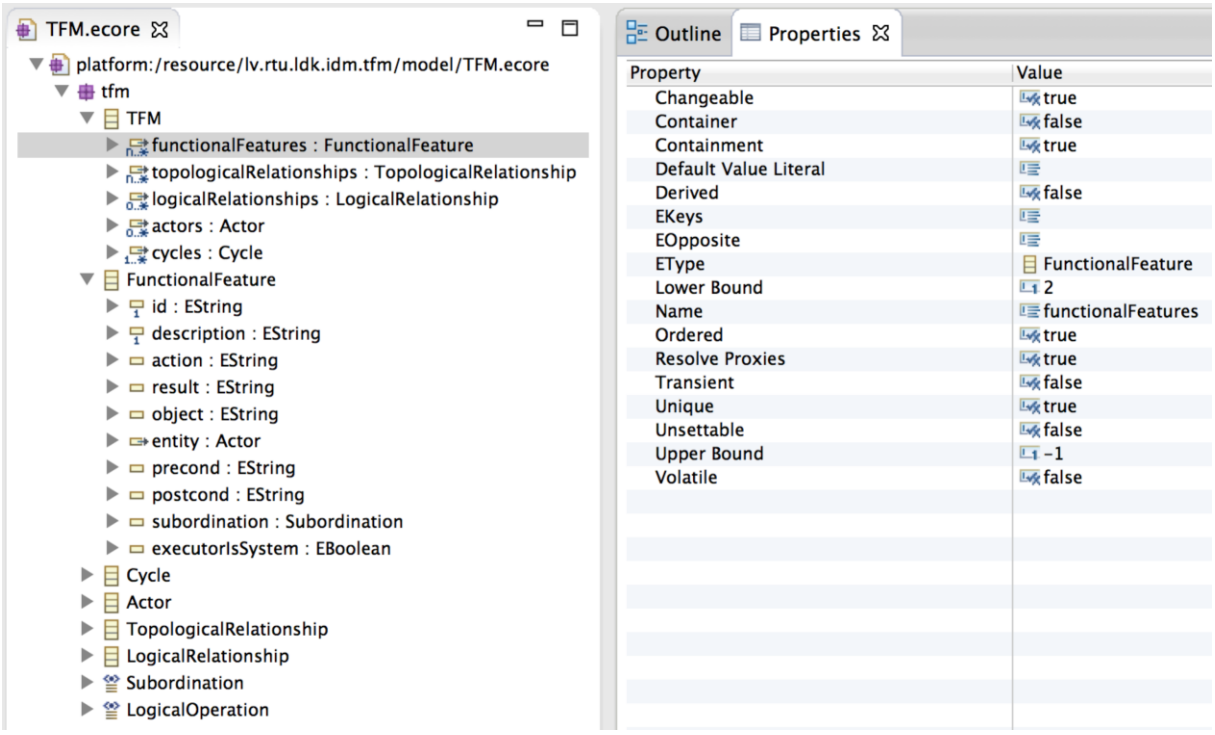


Figure 3.10. Ecore model for a TFM

Functional feature has the following attributes: id, description, action, result, object, subordination, pre-condition, post-condition and whether the executor is system. Each functional feature is related to a particular actor. Actor has an attribute description. Cycle has attributes order and whether it is the main cycle. Order is an identifier of the cycle. A cycle relates to at least 2 functional features. Topological relationship has an identifier, source topological feature and a target functional feature. Thus a topological relationship defines cause and effect relationship between

functional features. A logical relationship has an identifier, an operation, and related items. It defines a logical relationship between 2 or more topological relationships.

Figure 3.10 shows a MOF-compatible meta-model in raw form of Ecore for a TFM. This Ecore model exactly corresponds to the Ecore diagram in Figure 3.9. Full XMI of this model can be found in the appendix giving all details on the configuration. Once there is a meta-model defined in the form of Ecore, the next step is to develop a generation model for the EMF generator. Initial generation model can be generated automatically. Also for the TFM Editor the author uses the default values.

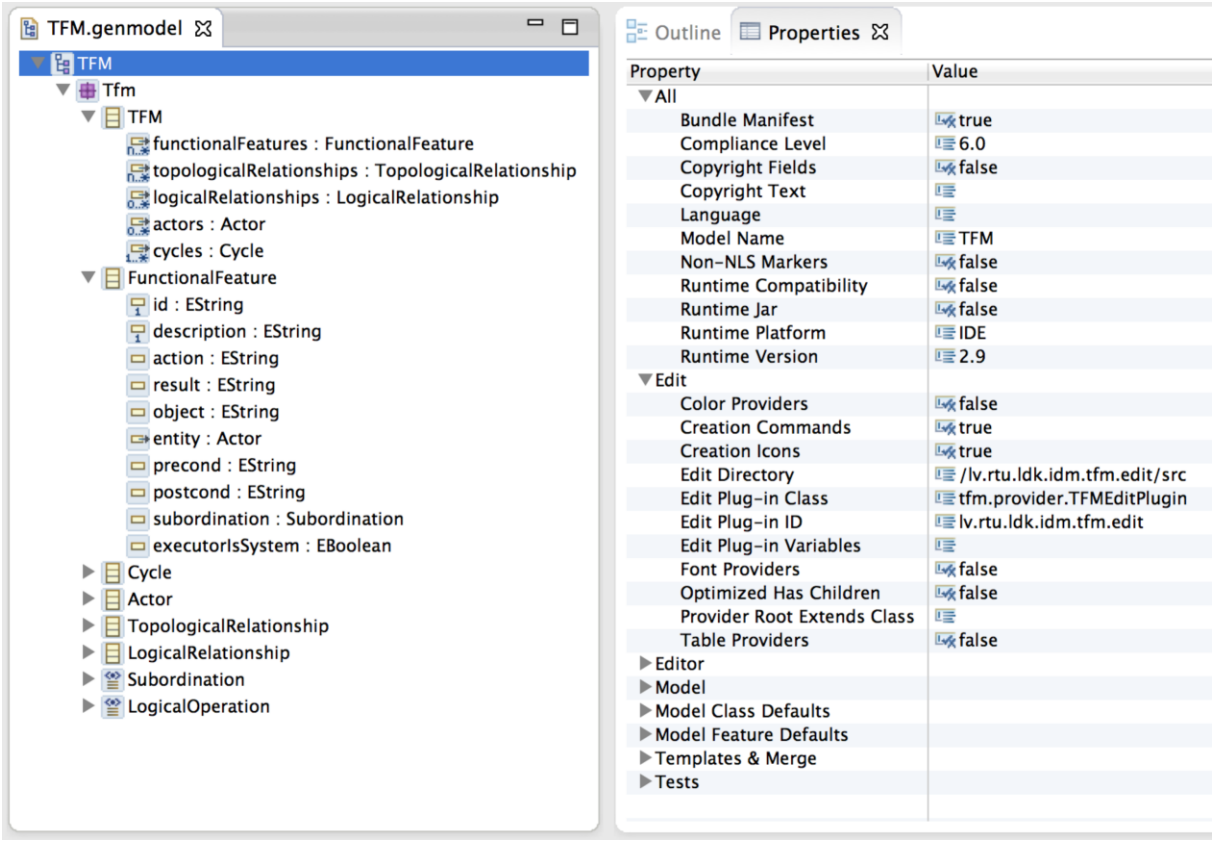


Figure 3.11. Generation model for TFM Editor

Figure 3.11 shows the generation model of EMF for TFM Editor. The root node of the model represents the model editor and the child node contains the meta-model element definitions for the tool. Full XMI of this model can be found in the appendix giving all details on the configuration. Using this generation model the initial source code for the TFM Editor has been generated. The classes and interfaces generated by EMF are marked with a special Java annotation “@generated”. It is possible to make changes to this source code, but in the case of TFM Editor the author uses the default

source code. The only changes for the generated plug-ins in the case of TFM Editor are the icons. For each of the elements, which are visible in the Use Cases Editor a special icon is provided to represent the element and help the user to understand the meaning of each element. Freely available icon set was reused and slightly modified for IDM toolset. It is possible to see the icons in Figure 3.12.

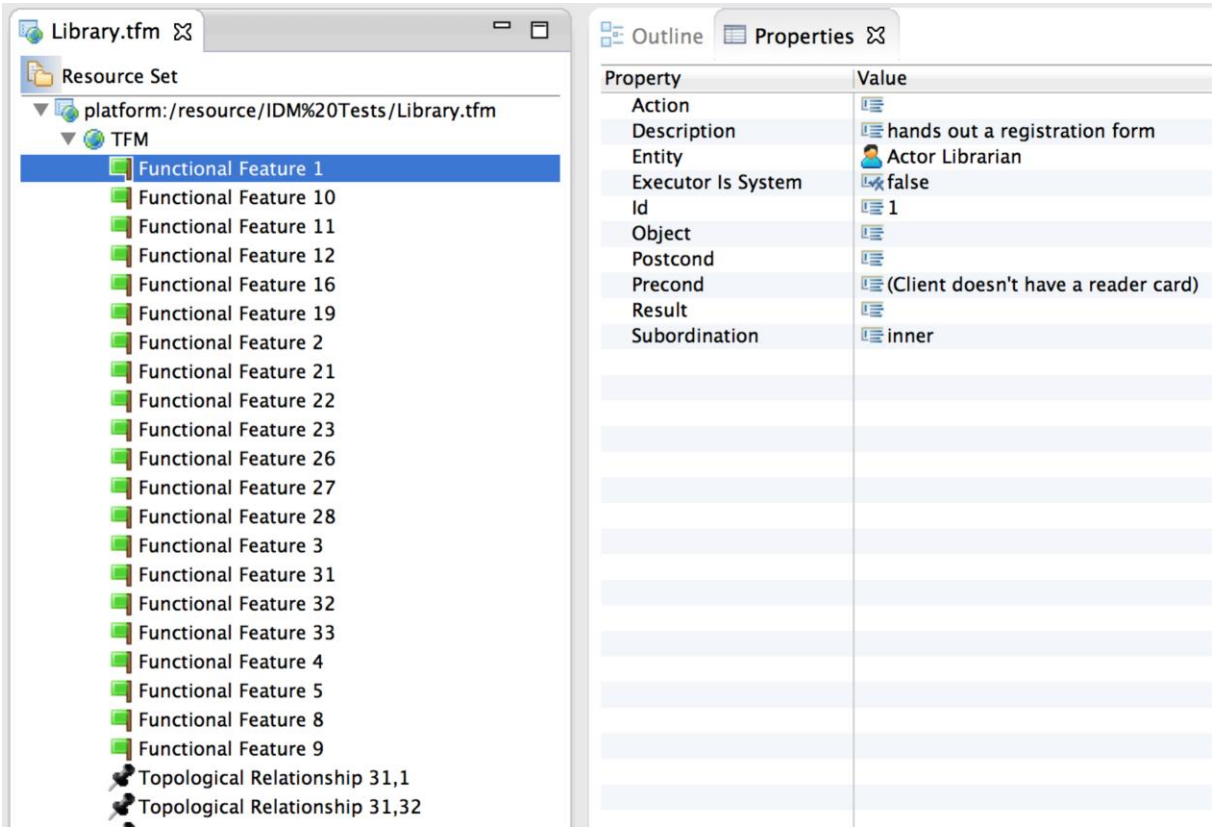


Figure 3.12. TFM Editor Tool

Figure 3.12 shows the resulting TFM Editor tool. On the left hand side there is a palette with the available objects for the diagram. In the middle there is a canvas with the diagram itself, and on the right hand side there is a properties pane and an outline view. The file extension of the TFM artifact is ".tfm". This tool allows the users to define a TFM – actors, functional features, topological relationships, cycles and logical relationships. To create an element the user need to right click on the TFM node and create a child element of the desired type. To set the attributes of an element the user must use the properties pane. The file extension of the Use Cases artifact is ".tfm". This file can also be opened with a text editor and then it is possible to see that underneath it has an XMI format.

3.5. TFM Diagram Tool

This sub-section talks about the TFM Diagram Tool implementation as an Eclipse plug-in done by the author. This tool is based on the Eclipse GMF. Author provides screenshots of the development snippets and configuration. The full source code and configuration can be found in the appendix.

With TFM Editor tool it is possible to edit the raw TFM model, but there should also be a possibility to initialize a diagram to see the graphical representation of the TFM. This is done via the TFM Diagram Tool, which is based on the Eclipse GMF. By developing special configuration models it is possible to generate the graphical modeling tool from the EMF based meta-model [72]. With both tools the TFM Editor and Diagram Tool it is possible construct/edit the TFM model. When editing the diagram (file extension ".tmf_diagram") also the underlying TFM model (file extension ".tfm") is changed by the diagram tool so that both artifacts are in sync. Eclipse EMF and GMF were used to achieve this goal.

3.5.1. GMF Workflow

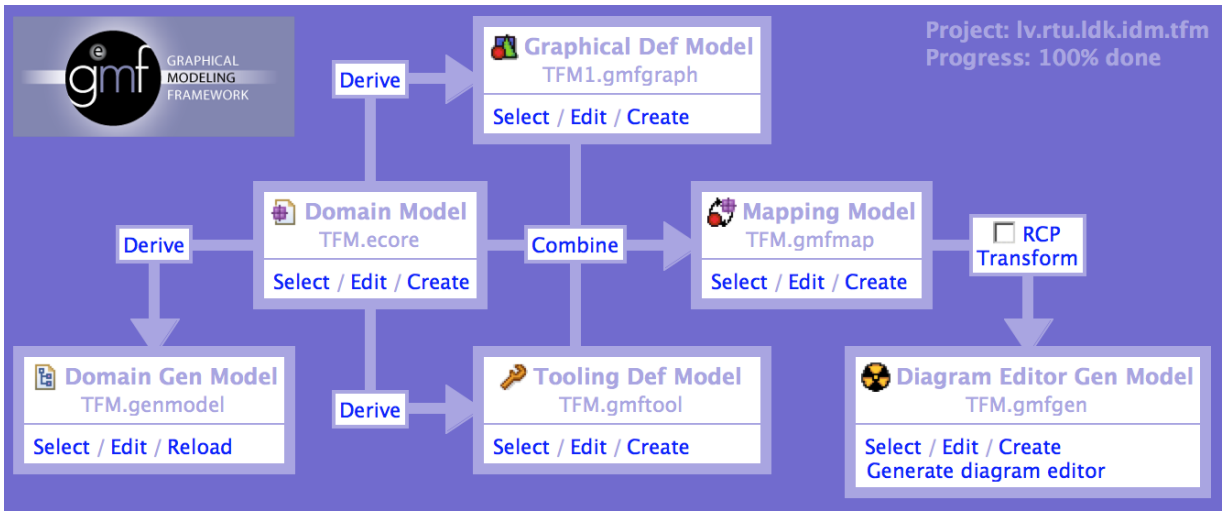


Figure 3.13. GMF Workflow

Figure 3.13 shows the GMF workflow for acquiring a diagram tool. This screenshot is taken from Eclipse GMF dashboard view, which is used to follow the workflow. It is possible to select, edit or create the corresponding models, or to do the

model transformations (derive, combine, transform). The diagram that illustrates the main components and models used during GMF-based tool development. A graphical definition model concept is core to GMF. This model contains information related to the graphical elements that will appear in a GEF-based runtime, but have no direct connection to the domain models for which they will provide representation and editing. An optional tooling definition model is used to design the palette and other periphery (menus, toolbars, etc.). It is expected that a graphical or tooling definition may work equally well for several domains [14]. For example, the UML class diagram has many counterparts, all of which are strikingly similar in their basic appearance and structure. A goal of GMF is to allow the graphical definition to be reused for several domains. Using a separate mapping model to link the graphical and tooling definitions to the selected domain model can do this.

After the appropriate mappings are defined, GMF provides a generator model to allow implementation details to be set for the tool generation. The production of an editor plug-in based on the generator model will target the diagram model. The runtime will bridge the diagram model and the domain model when a user is working with a diagram, and also provides for the persistence and synchronization of both. In the context of TFM Diagram Tool the domain model and domain generation model ("Domain Gen Model" in Figure 3.11) have been developed for the TFM Editor in section 3.5. For the diagram tool the author proceeds according to the GMF workflow creating the rest of the models.

3.5.2. Graphical Definition Model

Next step is developing the graphical definition model (file extension ".gmfgraph"). By using the Eclipse GMF dashboard it is possible to generate the initial model automatically from the domain model (meta-model) by using the derive functionality. Later it can be modified.

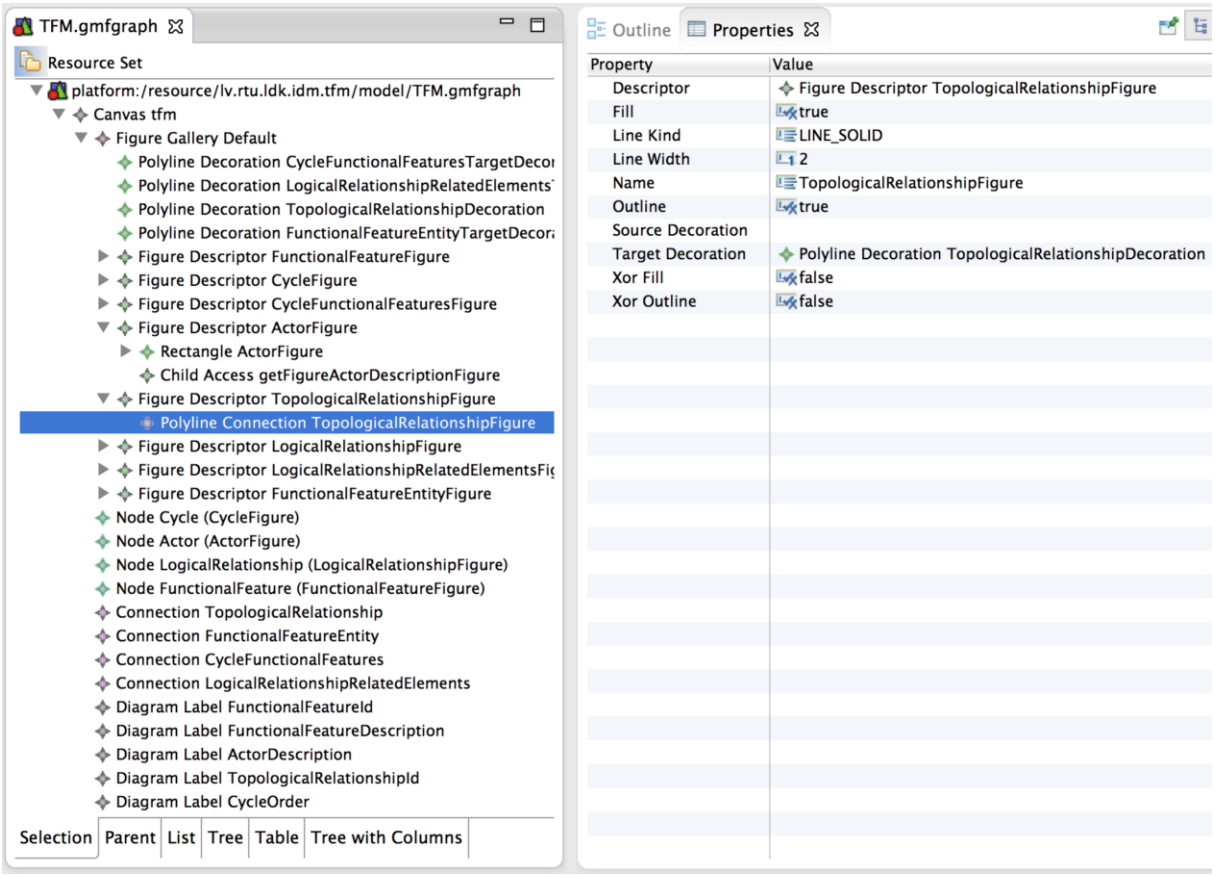


Figure 3.14. GMF Graphical Definition Model

Figure 3.14 shows the GMF graphical definition model, which is responsible for the nodes, connections, figures, decorations, and labels of the diagram. Full XMI of this model can be found in the appendix giving all details on the configuration. This model defines the mapping between diagram elements and the meta-model. First, it defines the nodes and connection of the diagram, and then the figures, decorations and labels, which impact how the nodes and connections are displayed to the user. There is a canvas at the root with a figure gallery containing basic rectangle, label, and polyline connection elements. These are used by corresponding node, diagram label, and connection elements to represent the elements from the domain model.

For the TFM diagram the graphical definition model defines the following:

- 4 nodes – cycle, actor, logical relationship and functional feature;
- 4 connections – topological relationship, functional feature entity (connecting a functional feature with an actor), cycle's functional feature (connecting a cycle with a functional feature), logical relationship's related elements (connecting a logical relationship with topological relationships);

- 6 diagram labels – functional feature identifier, functional feature description, actor description, topological relationship identifier, cycle order, logical relationship operation;
- Figure gallery.

The figure gallery consists of 8 figure descriptors and 4 polyline decorations:

- Functional feature figure, which has 3 child elements – rounded rectangle (defining the appearance of the figure in a form of a rounded rectangle with configuring background color, preferred size, layout and 2 labels), child access to the diagram label of functional feature's identifier and child access to the diagram label of functional feature's description;
- Cycle figure, which has 2 child elements – ellipse (defining the appearance of the figure in a form of a ellipse with configuring preferred size, layout and a label) and a child access to the diagram label of cycle order;
- Cycle functional feature figure, which has only 1 child element defining the appearance in a form of polyline;
- Actor figure, which has 2 child elements – rectangle (defining the appearance of the figure in a form of a rectangle with configuring background color, preferred size, layout and a label);
- Topological relationship figure, which has only 1 child element defining the appearance in a form of polyline;
- Logical relationship figure, which has 2 child elements – ellipse (defining the appearance of the figure in a form of a ellipse with configuring preferred size, layout and a label) and a child access to the diagram label of logical relationship operation;
- Logical relationship related elements figure, which has only 1 child element defining the appearance in a form of polyline;
- Functional feature entity figure, which has only 1 child element defining the appearance in a form of polyline.

3.5.3. Tooling Definition Model

Next step is developing the tooling definition model (file extension ".gmftool"). By using the Eclipse GMF dashboard it is possible to generate the initial model automatically from the meta-model (which is called "domain model" in context of GMF) by using the derive functionality. Later it can be modified.

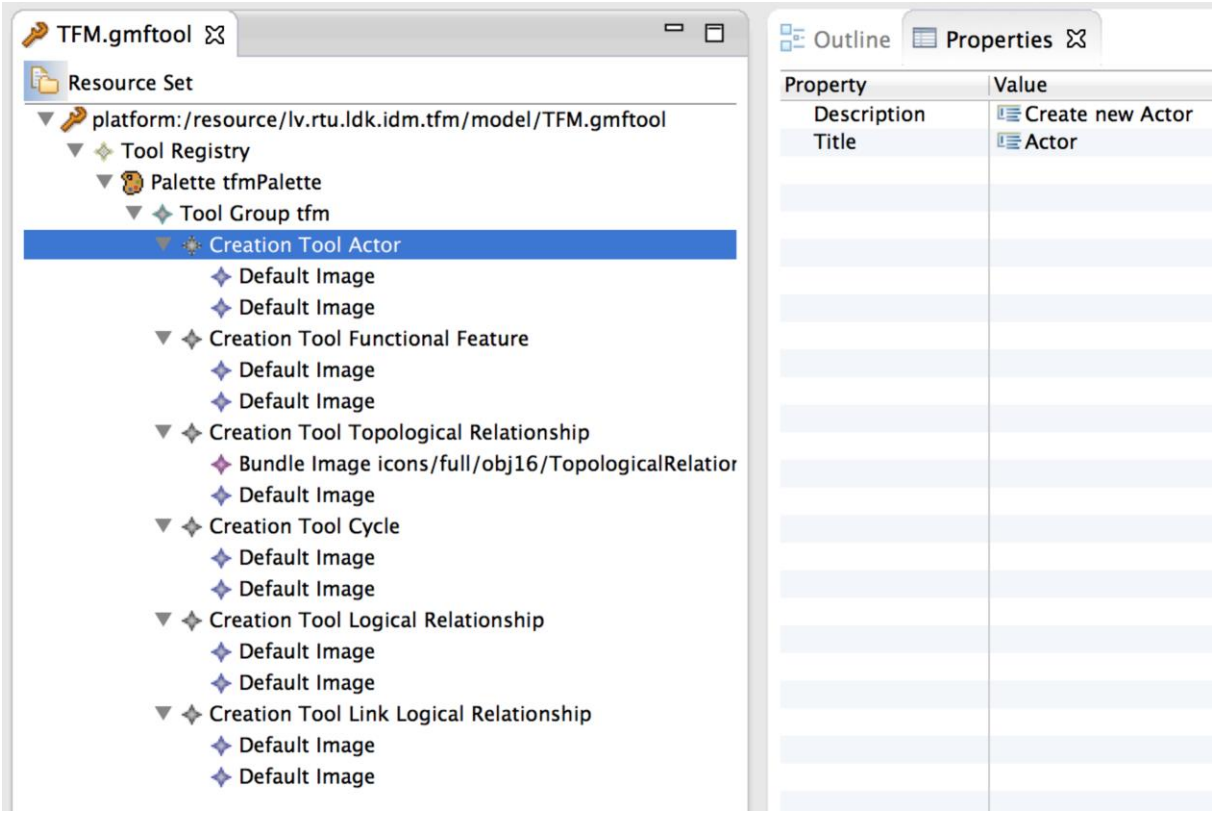


Figure 3.15. GMF Tooling Definition Model

Figure 3.15 shows the GMF tooling definition model, which is responsible for the palette of the diagram. Full XMI of this model can be found in the appendix giving all details on the configuration. The tooling definition model is responsible for the palette of the tool; defining what kind of diagram elements the user can create from the palette. As shown in Figure 3.15 there is a top-level tool registry element, under which there is a palette element. The palette contains a tool group with creation tool elements for nodes and links. For a TFM diagram the tooling definition model defines 6 creation tools – actor, functional feature, topological relationship, cycle, logical relationship and link logical relationship. Each of them has 2 images small and large. It is possible to set a custom image bundle or use the default image.

3.5.4. Mapping Model

Next step is developing the mapping model (file extension ".gmfmap"). By using the Eclipse GMF dashboard it is possible to generate the initial model automatically by using the combine functionality. This combines the graphical definition, tooling and domain models. Later the initial model can be modified. It is important in this step to make sure all previous models are correct, since until now there the models were standalone. In this step the 3 models are mapped and embedded within the mapping model and the elements get their meaning. For example, a domain element get's mapped to the corresponding diagram node and the creation tool.

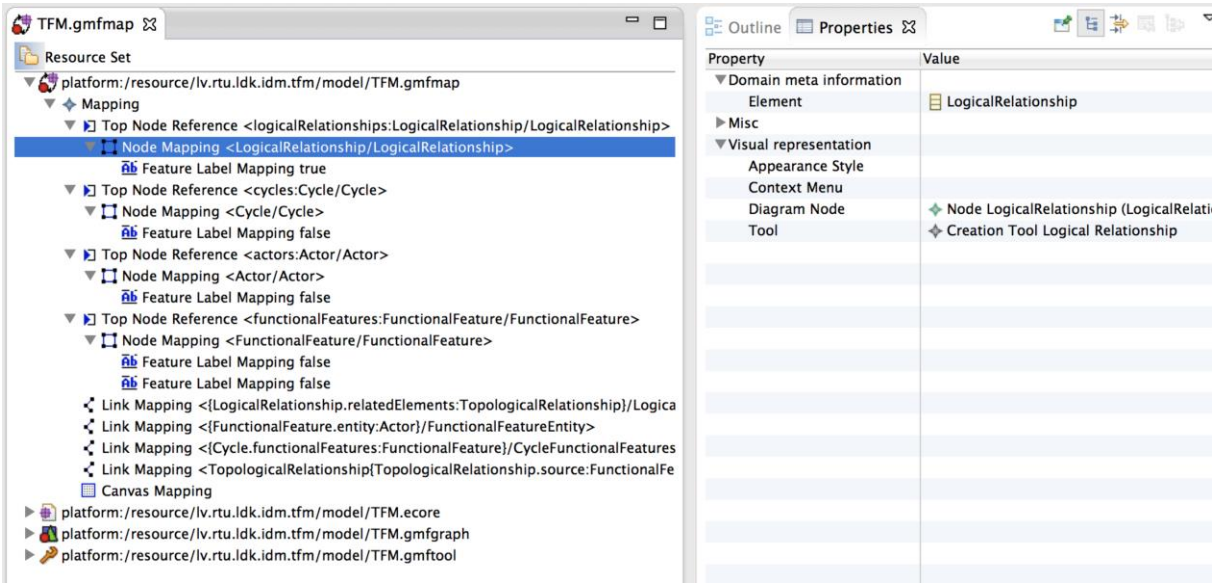


Figure 3.16. GMF Mapping Model

Figure 3.16 shows the GMF mapping model, which is responsible for mapping the domain model, graphical definition model and the tooling definition model. Full XMI of this model can be found in the appendix giving all details on the configuration. Using the mapping model, the actual nodes and connections defined in the graphical definition model, and tools defined in the tooling definition model are mapped with the domain model. This is done by defining the mapping model elements – top node reference, link mapping and canvas mapping.

For the TFM diagram the mapping model defines the following:

- 4 top node references – logical relationships, cycles, actors and functional features;
- 4 link mappings – logical relationship's related elements, functional feature's entity, cycle's functional feature and a topological relationship;
- Canvas mapping.

The top node references are defined as follows.

- For logical relationships the "logical relationship" element from the domain model is mapped to the diagram node "logical relationship figure" from the graphical definition model and the "logical relationship" creation tool from the tooling definition model. Also a feature label mapping is defined to map the diagram label with the domain element.
- For cycles the "cycle" element from the domain model is mapped to the diagram node "cycle figure" from the graphical definition model and the "cycle" creation tool from the tooling definition model. Also a feature label mapping is defined to map the diagram label with the domain element.
- For actors the "actor" element from the domain model is mapped to the diagram node "actor figure" from the graphical definition model and the "actor" creation tool from the tooling definition model. Again a feature label mapping is defined to map the diagram label with the domain element.
- For functional features the "functional feature" element from the domain model is mapped to the diagram node "functional feature figure" from the graphical definition model and the "functional feature" creation tool from the tooling definition model. For functional features there are 2 feature label mappings defined – one for the identifier and another for the description.

The link mappings are defined as follows.

- For logical relationship's related elements the "logical relationship related elements" element of type "topological relationship" from the domain model is mapped to the diagram link "logical relationship related

elements” from the graphical definition model and to the creation tool “link logical relationship” from the tooling definition model.

- For functional feature’s entity the “functional feature entity” element of type “actor” from the domain model is mapped to the diagram link “functional feature entity” from the graphical definition model. No creation tool is defined for this connection, because it will not be available in the palette. The user will be able to create the link directly in the diagram.
- For cycle’s functional feature the “cycle functional feature” element of type “functional feature” from the domain model is mapped to the diagram link “cycle functional feature” from the graphical definition model. No creation tool is defined for this connection, because it will not be available in the palette. The user will be able to create the link directly in the diagram.
- For a topological relationship there is a source and a target feature of the domain element. Source is the “topological relationship source” of type “functional feature” and target is “topological relationship target” of type “functional feature”. Thus there is a possibility to link 2 functional features with a topological relationship. Here also this domain element “topological relationship” is mapped to the diagram link “topological relationship” from the graphical definition model and the tool “topological relationship” from the tooling definition model.

3.5.5. Diagram Editor Generation Model

The last step is acquiring the diagram editor generation model (file extension “.gmfgen”). By using the Eclipse GMF dashboard it is possible to generate the initial model automatically from the mapping model by using the transform functionality. Later the initial model can be adjusted.

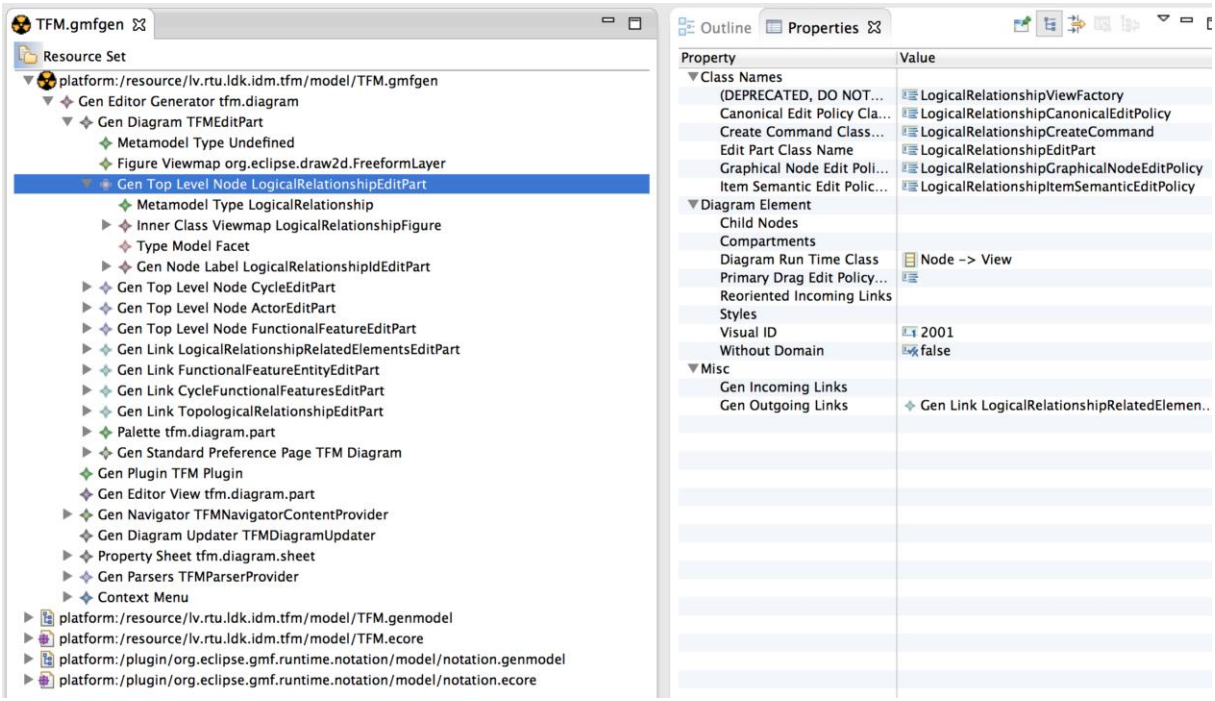


Figure 3.17. GMF Diagram Editor Generation Model

Figure 3.17 shows the GMF diagram editor generation model, which is responsible for the diagram editor source code generation. Full XMI of this model can be found in the appendix giving all details on the configuration. The generation model is used as the configuration for the generator. This model should not be used for conception of the editor, but rather to tweak the Java code that will be generated [14]. This model is used by GMF to generate diagramming code. The intention of this model is similar to the EMF generation model discussed in section 3.4.2. In the context of the TFM Diagram Tool the default transformation from the mapping model is sufficient. The default values of the diagram editor generation model are kept, since the business logic of the diagram tool has already been provided in the previous models, which lead to this point. The diagram editor generation model is sum of the work done previously expressed on a technical level for the code generation. Information on the tweaking options provided by this model can be found in the documentation [14].

To generate the source code it is necessary to execute the generation by choosing “Generate diagram code” on the root of the generation model. This will generate an Eclipse plug-in for the corresponding diagram editor.

3.5.6. Resulting Diagram Tool

After source code generation it is possible to run the diagram tool and try it out. It is dependent on these Eclipse plugins – TFM ("lv.rtu.ldk.idm.tfm"), TFM Edit ("lv.rtu.ldk.idm.tfm.edit"), TFM Editor ("lv.rtu.ldk.idm.tfm.editor"), TFM Tests ("lv.rtu.ldk.idm.tfm.tests") and TFM Diagram ("lv.rtu.ldk.idm.tfm.diagram").

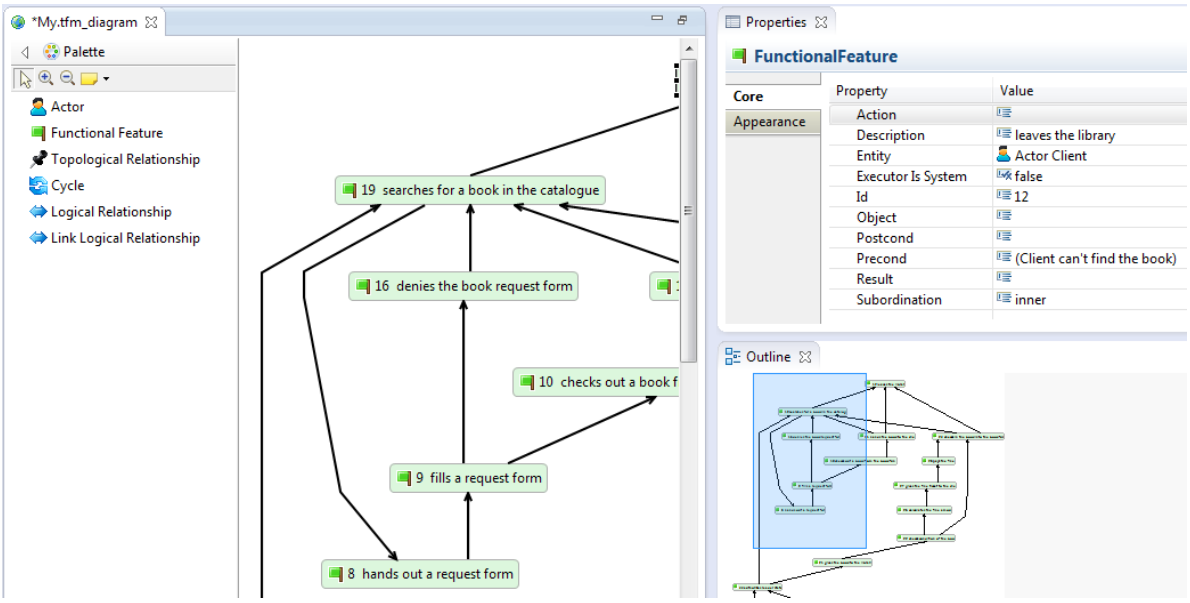


Figure 3.18. TFM Diagram Tool

Figure 3.18 shows the resulting TFM Diagram Tool, which was generated according to the GMF workflow and based on the model discussed earlier. On the left there is a palette with available tools for the user. In the center there is a canvas for the diagram. On the right there is the properties pane and the outline view. This tool allows the users to define a TFM in a diagram form – actors, functional features, topological relationships, cycles and logical relationships. To create the elements the user has to select the corresponding tool from the palette or right click on the canvas and select the desired element. Topological relationship has to be created between 2 functional features. Logical relationship has to be created between 2 or more topological relationships. The logical relationship link is used to link the logical relationship with the topological relationships. Cycle has to consist of 2 or more functional features. The attributes of the element can be set in the properties pane. The file extension of the TFM diagram artifact is ".tfm_diagram". This file can also be

opened with a text editor and then it is possible to see that underneath it has an XMI format.

3.6. Use Cases to TFM Transformation

This sub-section talks about the Use Cases to TFM model-to-model transformation implementation as an Eclipse plug-in done by the author. This tool is based on Eclipse QVTo and Stanford Parser. Author provides screenshots of the development snippets and configuration. The full source code and configuration can be found in the appendix.

As described previously in the section 2 “The IDM Modeling Approach” there is no need to construct the TFM from scratch and the initial model can be generated automatically from the Use Cases model. After that the user can modify the TFM model and add elements if necessary. The model transformation is handled by the Use Cases to TFM Transformation tool, which is based on the QVTo model transformation language. The transformation depends on natural language processing so we use the Stanford Parser Java library [TMP-16]. This is integrated into the transformation tool via the QVT-BlackBox extension mechanisms suggested in the QVT standard and also Eclipse QVTo supports this approach [42]. The model transformation and the black box are packaged into a single Eclipse plug-in and form the Use Cases to TFM Transformation tool. This also includes some Eclipse PDE developments to integrate with the Use Cases Editor tool.

3.6.1. Model-to-model Transformation

*UseCases2TFM.qvto

```
56
57      //Create Functional Features
58      self.allSubobjectsOfKind(SingleEvent)[UseCases::SingleEvent]->forEach(step) {
59          resolvedDescription := parseStepForDescription(step.description);
60          resolvedEnity := parseStepForEntity(step.description);
61
62          //If step is referenced no need for a seperate Functional Feature
63          if (step.reference.oclIsUndefined()) then {
64
65              //Get referenced steps for conditions
66              referencedSteps := step.getReferencedSteps(self);
67
68              //Create functional features
69              functionalFeatures += step.map step2functionalFeature(actors,
70                  resolvedEnity, resolvedDescription, referencedSteps);
71
72          } endif;
73      };
74
```

Figure 3.19. Use Cases to TFM Transformation

Figure 3.19 shows the model transformation "UseCases2TFM.qvto" written in QVTo. This is a small snippet from the entire model transformation code, which deals with generating functional features. The full source code can be found in the appendix. The particular snippet deals with one of transformation's main tasks – creation of Functional Features in the TFM model. First the transformation loops over all steps of the Use Cases model. It is also shown how the description and the entity for the Functional Feature are resolved from Use Case steps with methods – "parseStepForDescription" and "parseStepForEntity", which in order call the Stanford Parser's library to find the noun and verb phrases. These methods are provided by the QVT-BlackBox library implemented in Java for this transformation developed by the author. Next the transformation checks if the step is referenced, and if not then it prepares to create a Functional Feature. The model transformation needs to know the steps, which references the current step so that it is possible to merge the pre-conditions and post-conditions (in case there are any). This is done by QVTo mapping and helper functions. This is a small part of the model transformation code, but in similar fashion we deal with Actors, Topological Relationships, etc. This model transformation from Use Cases to TFM enables the IDM toolset to fulfill its promise and automatically transform the domain knowledge expresses in Use Cases into a

domain model in a form of a TFM. Thus the user acquires a graphical representation of a working business system by defining it with means a common enterprise standard – Use Cases.

3.6.2. Natural Language Processing

In order to incorporate Natural Language Processing (NLP) for the IDM Toolset, the author uses the concept of QVT-BlackBox. It is a way to include some custom Java coding for QVT model transformations. There is a Utilities Library class defined as a QVT-BlackBox and used within the model transformation. The Utilities Library in order will use Parser Tools class, which imports and used the Stanford Parser library.

UtilitiesLibrary.java | ParserTools.java

```
1  package lv.rtu.ldk.idm.usecases2tfm.blackbox;
2
3  public class UtilitiesLibrary
4  {
5    private static ParserTools parserTools = new ParserTools();
6
7    public static String parseStepForEntity(String stepDesciption)
8    {
9      return parserTools.getEntity(stepDesciption);
10   }
11
12   public static String parseStepForDescription(String stepDesciption) {
13     return parserTools.getDescription(stepDesciption);
14   }
15 }
```

Figure 3.20. Utilities Library for IDM Toolset

Figure 3.20 shows the "UtilitiesLibrary.java" class of the IDM Toolset's package "lv.rtu.ldk.idm.usecases2tfm.blackbox". Utilities Library is developed according to the facade software design pattern to provide a simplified interface to a larger body of code. Thus, it makes the Parser Tools class easier to use, understand and test, since the facade has convenient methods for common tasks. The Utilities Library has a private attribute "parserTools" and 2 public methods:

- "parseStepForEntity" – to parse a given Use Case Step of type String and return the Entity for a TFM'S Functional Feature of type String;

- “parseStepForDescription” – to parse a given Use Case Step of type String and return the Description for a TFM’S Functional Feature of type String.

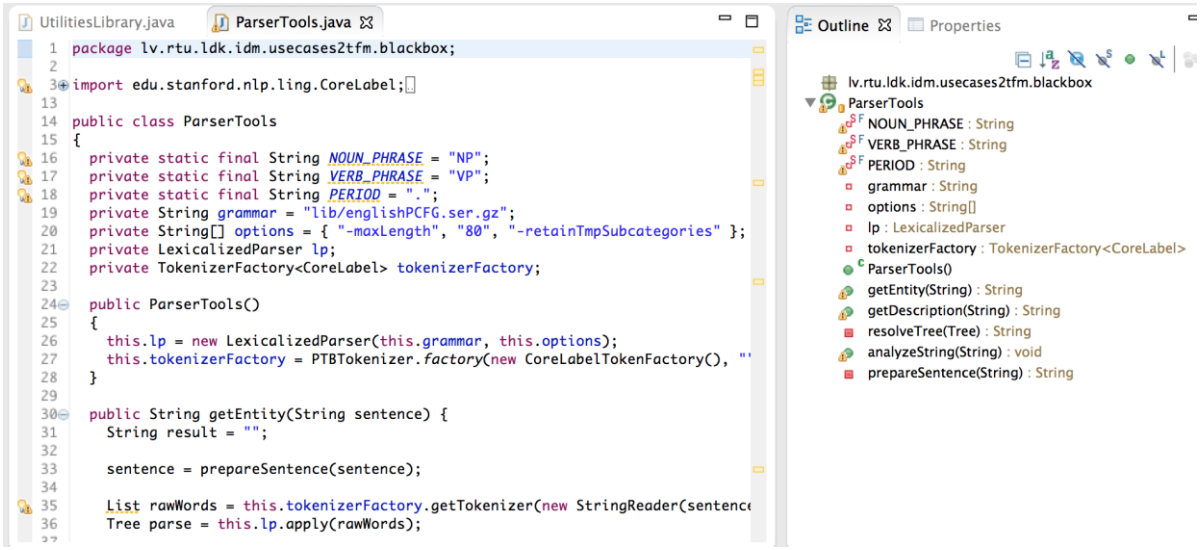


Figure 3.21. Parser Tools for IDM Toolset

Figure 3.21 shows the “ParserTools.java” class for IDM Toolset’s package “lv.rtu.ldk.idm.usecases2tfm.blackbox”. The full source code can be found in the appendix. The Parser Tools class has a constructor, which initializes the Lexical Parser for English and the Tokenizer Factory. Theoretically, here a grammar library for another language could be used, if that language has constructs of verb and noun phrases. However, the author has not tested this. IDM Toolset supports only sentences in English at this moment. The constructor is called when initializing the class in the Utilities Library.

The main methods of the Parser Tools class are the following.

- “getEntity” – to parse a given string and return the noun phrase to be used later as the Entity. First the string is tokenized and then the Lexical Parser is used to acquire a parse tree. Then the there is a loop over the tree to find the noun phrase of the sentence, which is returned as a string.
- “getDescription” – to parse a given string and return the verb phrase to be used later as the Description. First the string is tokenized and then the Lexical Parser is used to acquire a parse tree. Then the there is a loop over the tree to find the noun phrase of the sentence, which is returned as a string.

3.6.3. Toolset Integration

To run the model transformation the described in previous section a user would define a new Eclipse Run Configuration with the Operational QVT Interpreter and point to the "UseCases2TFM.qvto" file. This is the default approach supported by the Eclipse QVTo. Since this adds an additional complexity to the usability of the IDM Toolset, the author has developed the toolset integration part discussed in this section to avoid the additional step.

The idea behind the toolset integration is that the user should only right click the Use Cases model file and choose the option "Transform to TFM" to trigger the model transformation. To implement this the author used the Eclipse Plugin Development Environment (PDE).

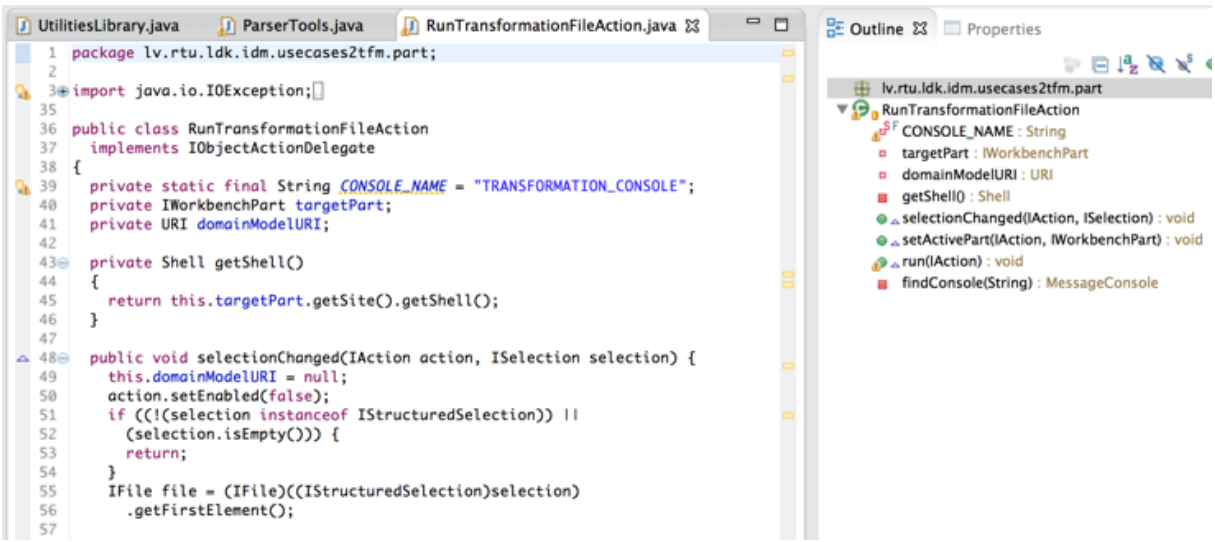


Figure 3.22. Run Transformation File Action

Figure 3.22 shows the "RunTransformationFileAction.java" class for IDM Toolset's package "lv.rtu.ldk.idm.usecases2tfm.part". The full source code can be found in the appendix. The Run Transformation File Action class provides an action to the Eclipse Use Cases to TFM plugin. This is done using the manifest of the Eclipse plugin (see Figure 3.23). The main method of the action class is "run", which is developed by the author to do the following:

- Contain the path to the "UseCases2TFM.qvto" file within the plugin;
- Do the setup for the Operational QVT Interpreter;

- Set the resulting model's file extension to ".tfm";
- Run the Use Cases to TFM model transformation.

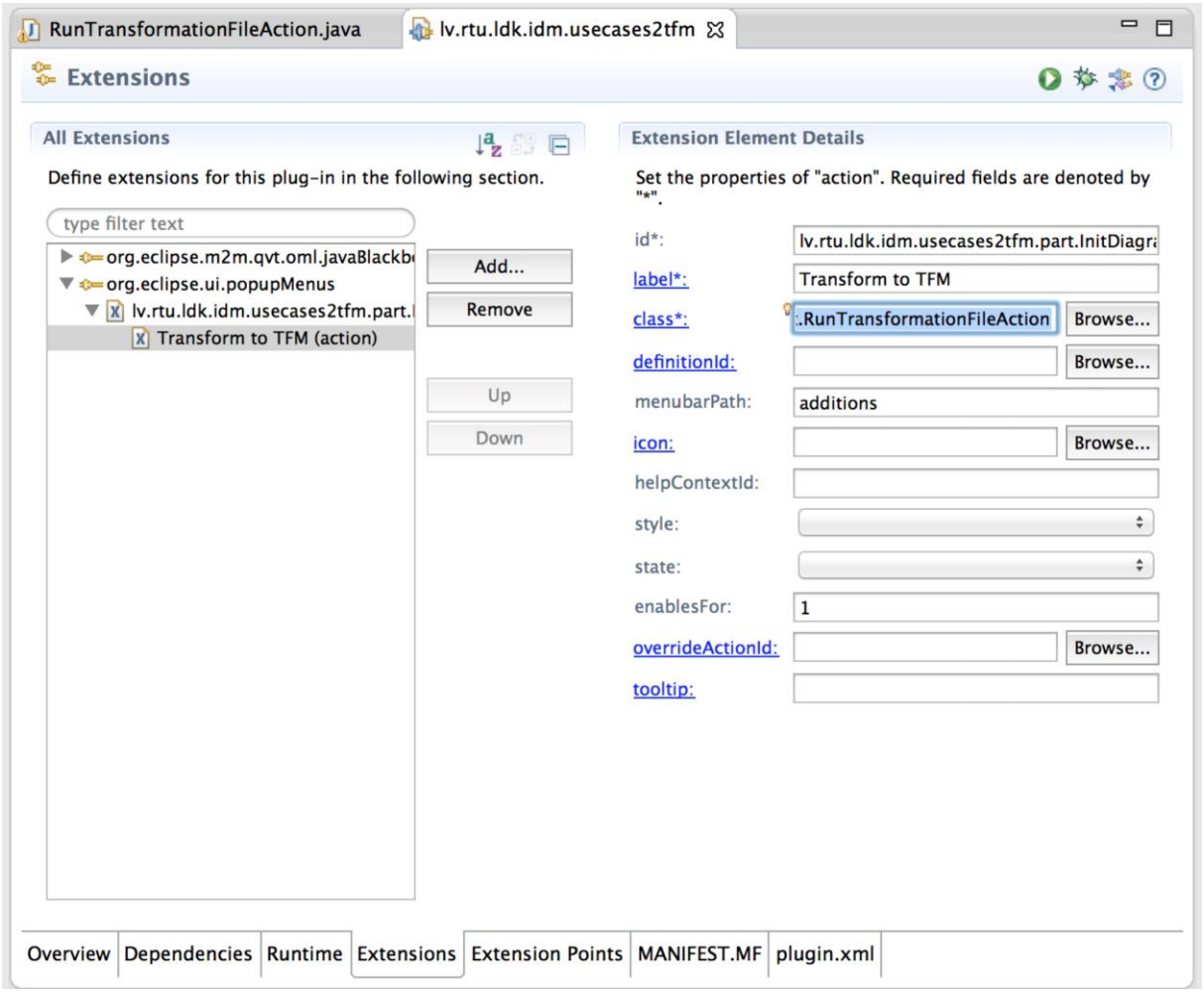


Figure 3.23. Use Cases to TFM Plugin Manifest

Figure 3.23 shows the "MANIFEST.MF" class for IDM Toolset's plugin "lv.rtu.ldk.idm.usecases2tfm". The full configuration can be found in the Appendix 8. In the Eclipse plugin manifest configuration the author uses Eclipse UI Popup Menus extension "org.eclipse.ui.popupMenus" to introduce additional functionality in the popup menu. There is a setting to attach this functionality only for files with the given extension – ".usecases". After that it is possible to set the label and class of the action to be used, which is done correspondingly. Label for the action is set to be "Tranform to TFM", and the class of the action is set to "lv.rtu.ldk.idm.usecases2tfm.part.RunTransformationFileAction".

3.6.4. Resulting Model Transformation Tool

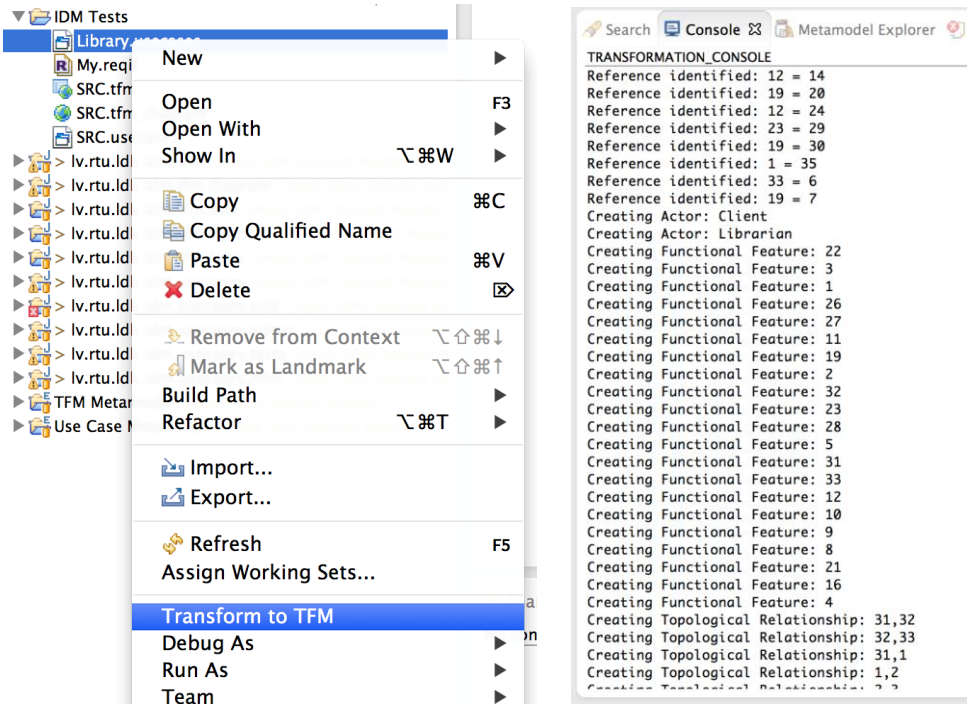


Figure 3.24. Use Cases to TFM Transformation Resulting Tool

Figure 3.24 shows the resulting model transformation tool. On the left side there is the popup menu shown and the option to execute the transformation. On the right side the console output after executing the transformation is shown. To execute the transformation a user has to right click on a file with extension ".usecases" and select the option "Transform to TFM". After that the user can check the console to see the log of the model transformation. This shows references with identifiers, Actors created, Functional Features created and Topological Relationships created. In the same project a new file will appear after the model transformation with the extension ".tfm". This is the TFM model's file.

3.7. Summary

This section introduced the supporting toolset of the Integrated Domain Modeling (IDM) approach, discussing the architecture, used technologies, implementation, model transformation, and application. The IDM toolset allows defining the business processes with Use Cases, validating them against the

corresponding Ontology and then generating the domain model automatically in the form of a Topological Functioning Model (TFM). This toolset enables the system analyst to acquire a formal domain model. This way it is possible to validate the business processes (with Use Cases and TFM) at the beginning of the software development, by ensuring that they correspond to the Ontology, by defining the scope based on inputs and outputs, and by also checking the functioning cycles of the processes (represented in TFM). Exploiting the domain model acquired by the IDM toolset the system analyst together with the business people can validate the business processes before the actual software developments start. Thus it is possible to make sure that the business processes correspond to the business domain. This way debugging can be done in the domain model even before any line of code has been written.

A user guide has been created for the IDM toolset in Appendix 9 and students have tested the toolset for acquiring a TFM for their domains, and 10 of them have given feedback on the toolset in a survey in Appendix 10. The approbation of the toolset by the author for a real software development projects follows in Section 4.

4. APPROBATION OF THE INTEGRATED DOMAIN MODELING APPROACH

This section provides a case study of applying the IDM approach and toolset to a Subscription Commerce business case. The case study is based on a project done by Pearl Consulting AS, Norway (SIA "Pearl Baltic Labs" in Riga, Latvia) for a customer who is in subscription commerce business, which operates in Norway, Sweden, Denmark, Finland and United Kingdom. The name of the particular Pearl Consulting AS customer will not be mentioned, but referred to as SCB – Subscription Commerce Business. The integrated solution for SCB consists of the following main components: ERP, CRM, BI and e-commerce. It is based on the following SAP products: SAP ECC, SAP CRM, SAP BW, SAP PI and Hybris. SAP ECC is used for logistics and financial processes. SAP CRM is used for campaign and customer center related processes. SAP BW is used for all business intelligence reports and dashboards. SAP PI is used for all interfaces between SAP, Hybris and 3rd party systems. Hybris is the central e-commerce component used for managing products and publishing products in the web shop, which is the main channel for customer subscriptions and marketing activities. The solution is integrated and some data is replicated between systems to handle different processes in each of the components. The author was involved in all stages of the project and was specifically responsible for the e-commerce component. Thus the main focus of the case study is the e-commerce business processes, although also CRM and ERP processes have been analyzed on a high level to put the e-commerce processes in perspective.

The software product choice for the solution was made by SCB at the very beginning of the project based on the proposal of Pearl Consulting AS. Still domain modeling is an essential part for understanding the domain AS-IS (as the business operates at the time of project beginning) and TO-BE (as the business should operate after the implementation project). For this particular project AS-IS and TO-BE domain models are very similar since one of the goals was to keep the existing processes. Therefor no effort was put in for conducting an explicit AS-IS analysis, but rather the processes were discussed and a TO-BE analysis took place. This case study focuses on the TO-BE domain model of the Subscription Commerce Business (SCB), and

provides explicit artifacts of the TO-BE domain model. SAP products, in addition to a comprehensive set of application development tools, also provide many out-of-the-box business processes. The business processes can be configured to match the required TO-BE SCB processes. To be sure which processes should be implemented and how should they be configured, a thorough blueprinting phase took place, during which the business domain was analyzed, and a final domain model according to the IDM approach provided. The IDM toolset was used for developing the Use Cases and to acquire the TFM.

The IDM approach is going to be used for acquiring a domain model for SCB and afterwards the author is going to analyze this case study, and compare the results to TFM4MDA (the previous approach of acquiring a TFM) to show the benefits of using IDM.

4.1. Domain

SCB offers their customers to subscribe to different kinds of products to be mailed to the address they want on a periodic basis. A product consists of several articles, which are shipped as bundles. The subscriber pays a fee per shipment of a bundle of the chosen product. In addition the customer can receive sample and gift bundles free of charge. SCB provides a variety of products in the following categories: Supplements, Skin Care, Barbering, Textiles and Music. The customer can unsubscribe from products, if he doesn't want to receive the products anymore, or he can change the frequency and details of deliveries. Campaigns and target groups are created in CRM based on master data stored in ERP and CRM. However the target groups can be used also in e-commerce to offer different web content based on the customer and propose discounts. Thus target groups will be replicated to e-commerce from CRM.

A central part of the SCB business process is product management, which consists of managing articles, bundles, products and adding subscription rules to the bundles. To clarify the product structure, author gives an example. There is a product a customer can subscribe to named "Wilkinson Hydro 5". It consists of several bundles – sample bundle, subscription bundle and periodic bundle. The sample bundle consists

of 2 articles – Wilkinson Hydro 5 razor and Wilkinson Foam. In addition each bundle has a corresponding packaging article and a shipping article. These define the package material and shipping method. The sample bundle will be shipped to the customer at subscription or re-subscription (this is the subscription rule of the sample bundle). Subscription bundle of this product consists of single article – Wilkinson Blades, and will be shipped to the customer once every 3 months. This bundle Periodic bundle of this product consists of another article – Wilkinson Hydro 5 razor, and will be shipped every 10^{th} delivery of the subscription bundle. This is to replace the razor handle for the customer with a new one. Articles, bundles and products are defined using the e-commerce component and then replicated to CRM where they will be used in campaigns and budget planning, as well as replicated to ERP for logistics and finances.

4.1.1. Article Management

Article is an item available for product bundles. Articles are grouped into multi-level groups – categories. When article is created, it is assigned to a particular category. Packaging articles, which are used for logistics have master data definition similar to regular articles, e.g., dimensions, weight, volume, which have significant importance for packaging and shipping process. Shipping article defines the method of shipping, for example, Norwegian Post. The product manager creates an article based on template using the web user interface in the e-commerce component called "Product Cockpit". The following article basic attributes need to be set: name, description, weight, height, volume and price. Article needs to be added to one of the categories. In addition, articles also have variant attributes – size, style and shape. Approved articles can be used to create bundles.

4.1.2. Bundle Management

Bundle is a set of articles, including a packaging article and shipping article. A bundle also has a subscription rule, which defines the shipment of the bundle. The product manager creates a bundle based on a template in the web user interface of the

e-commerce component – Product Cockpit. Single articles or multiple articles can be added to the bundle. A single packaging article and, as well as, a single shipping article is mandatory for a bundle. When a bundle is approved it can be used for a product.

4.1.3. Product Management

Product is a set of bundles. It has to contain at least 1 bundle. The product manager creates a product based on a template in the web user interface of the e-commerce component – Product Cockpit. Also up selling options can be defined for a product, which would be reference to another product. Up selling means, that during subscription an alternative product can be suggested to the customer. When the product is constructed it can be published to the frontend of the e-commerce component, and thus be available for subscription by customers. During subscription the customer can select the possible article variants of the subscription bundles (if there are any). For example, for men's socks the customer could choose a color of the articles. The customer can later log into the e-commerce and change the frequency of the subscription or cancel certain deliveries; or follow up on the status of his deliveries. When a customer has confirmed the subscription it will be replicated to CRM and ERP as a contract.

4.1.4. Customer Subscription

Customer subscription happens through the website of SCB and start by entering the URL in the web browser. In the website the customer chooses products from the available categories through the navigation of the website. When the customer decides to subscribe to a particular product he needs to make a choice according to available article variants, enter his customer details, accept the terms and conditions and approve his subscription. For some of the products there may be article variants, for example, socks have a size and a color. There is also a duplicate check and validation to identify an existing customer. The e-commerce system Hybris will register new subscriptions between the customer and SCB, and these subscriptions will

automatically be replicated to ERP, where the Bundle Delivery can start. The customer will then receive an invoice together with his delivery.

4.1.5. Bundle Delivery

Bundle Delivery process happens in the ERP system. Based on customer subscriptions there is a contract per subscription in the ERP, thus these are collected to create and update the delivery schedule. Delivery schedule consists of planned shipments to the customers. Before approving the shipment the Warehouse Manager has the possibility to manually change some delivery details and add marketing materials. The invoice for the delivery will be part of the bundle and generated automatically base on the subscription. Thus the bundles are shipped to the customer.

4.1.6. Batch Purchase

Batch Purchase process is responsible for making sure that there are enough articles in the stock. This is handled in the ERP and controlled by the Article Manager. Article Manager can see the projected stock for the next planning period based on the delivery schedule and subscriptions. When the projected stock indicates a shortage the Article Manager based on the stock level an available price will create a batch purchase order. Thus the stock level is kept on the necessary level.

4.2. Scope of the Case Study

Although the project scope includes also ERP, CRM and BI systems this case study will focus on the e-commerce part of the business domain. In previous section the author describes the processes handled by e-commerce component – Article Management, Bundle Management, Product Management and Customer Subscription, but also mentions the interfaces for other systems – Bundle Delivery and Batch Purchase. In this case study the domain model will show the processes and structure of the e-commerce components, but in addition the author will also note integration points and interfaces to other systems. The author focuses on the customer interaction

with the SCB, which happened via the e-commerce component. Bundle Delivery and Batch Purchase is presented on a very high level not going into details, because this is the focus of the ERP system and the warehouse processes.

4.3. Developing the Ontology

This sub-section shows the development of the Ontology for the Subscription Commerce Business. Protégé tool is used for this purpose. Although the Ontology and Use Cases can be built in parallel, it makes sense to start with at least a basic set of core terms defined in the Ontology before starting to creating the Use Cases. For this SCB case the author did start with the Ontology and specifically the class hierarchy.

Figure 4.1. Ontology for SBC

Developing an Ontology includes: 1) defining classes in the Ontology; 2) arranging the classes in a taxonomic hierarchy; 3) defining properties and describing the relationships between classes; 4) defining individuals. Recognizing a satisfactory scope for the domain is not easy. It will not always be possible to capture everything on the first try, but as mentioned before this has to be an iterative process. The ability to modify the Ontology is crucial, because the scope of the domain can also change. Figure 4.1 shows class hierarchy of the Ontology for SCB e-commerce component (Protégé tool was used for developing this Ontology). The nodes of the graph are the classes and the edges are oriented hierarchical relationships between classes. There are some classes with equivalence defined, and then the hierarchical relationship goes both ways. For example, class "Person" is equivalent to class "User" in the context of this business domain. Ontology will be used to validate the Use Cases defined in the next section. The validation is described in section 4.5.

4.4. Developing the Use Cases

This sub-section shows the development of the Use Cases for the Product Subscription Business with IDM Use Cases Editor. The following 6 Use Cases were identified for SCB TO-BE process:

- Article management – creating and modifying articles in the e-commerce Product Cockpit;
- Bundle management – creating and modifying bundles in the e-commerce Product Cockpit;
- Product management – creating and modifying products in the e-commerce Product Cockpit;
- Customer subscription – subscribing to a product in the e-commerce website;
- Bundle delivery – managing the shipment of bundles to the customers in ERP;
- Batch purchase – managing the purchase of articles in ERP;

There could be more Use Cases in the full scope of the project, but since the focus of the case study is e-commerce, these are the most relevant ones. Each of the Use Cases are handled by one of the main system components – e-commerce, ERP or CRM. Also for each of the Use Cases there is a responsible actor from the SCB side

and from the system component's side. There are 7 actors defined in the Use Cases – customer, article manager, product manager, warehouse manager, hybris (e-commerce component), ERP, CRM. Each of the Use Cases needs to be analyzed in detail and defined with the IDM Use Cases Editor tool. With this tool it is possible to define the actors, conditions, main scenario steps and alternative scenario steps of the Use Cases. The user can use the tool while discussing the business process with the stakeholders to record the processes, or he can do this after the discussions. During the Use Case main scenario step definition, in addition to description of the step the user should capture the pre-conditions, post-conditions, extensions and sub-variations of the steps.

4.4.1. Article Management Use Case

Article Management deals with creating the articles to be later used in bundles for products. The responsible person for this is the Product Manager. To create articles the e-commerce platform's Hybris Product Cockpit is used, and thus the second actor of this Use Case is Hybris. In Figure 4.2 a screenshot of the IDM Use Cases tool is shown with the Article Management Use Case "UC-1" expanded. The actors involved in this Use Case are Product Manager and Hybris. There is a main scenario and 2 sub variations (alternative scenarios of type sub variation) – "A-1.1-Edit" and "A-1.2-ArticleVariant". The main scenario goes through the process of Product Manager logging into the Hybris Product Cockpit and creating a new article with the necessary attributes. The first sub variation is for the scenario of editing or modifying an existing article. The second sub variation is for the scenario of creating article variants with a base article. Alternative scenarios specify the alternative flow of events. For example, sub variation "A-1.1-Edit" references step "S-1.0.7", which states "Product Manager chooses to create a new article". Since the alternative scenario is of type sub variation (not the other possible type, i.e., extension) the new flow of events will come in parallel to the referenced step.

The alternative scenario ends with step "S-1.1.3", which states "Hybris shows the Product Cockpit". At this point it would be possible to specify that it is a reference to step "S-1.0.5" from main scenario of the Use Case, but it is not necessary since the IDM toolset will do it automatically during the model-to-model transformation. Also

the main scenario ends with step “S-1.0.15”, which states “Hybris shows the Product Cockpit”, thus creating a cycle for the Product Manager to be able to work on the next article or choose another action (going into other Use Cases of the Product Cockpit). Conditions need to be created and set as pre-conditions and post-conditions where applicable. In this example, step “S-1.0.1” stating “Product Manager opens the Product Cockpit” has a pre-condition “User is within SCB VPN”. Thus meaning that the Product Manager needs to be within Virtual Private Network (VPN) to access the Product Cockpit (to at least see the login page).

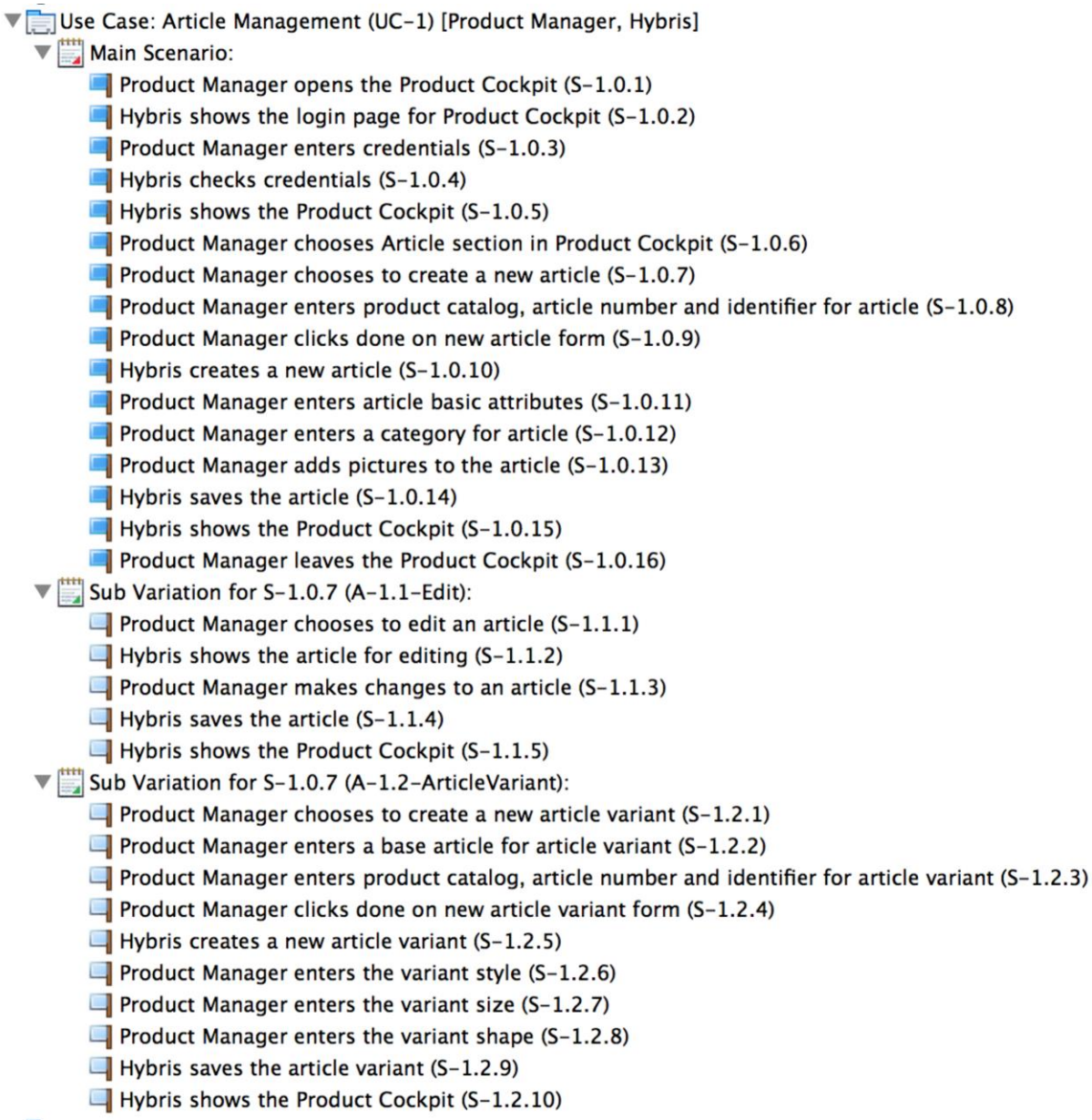


Figure 4.2. Article Management Use Case defined with IDM toolset

Another example of conditions is for step “S-1.0.5” stating “Hybris shows the Product Cockpit”. This step has a pre-condition “Credentials are confirmed”. Thus Product Manager will be able to see and use the Product Cockpit (after credentials are confirmed).

4.4.2. Bundle Management Use Case

In a similar way it is necessary to analyze the Bundle Management process and create Use Case with the IDM toolset.

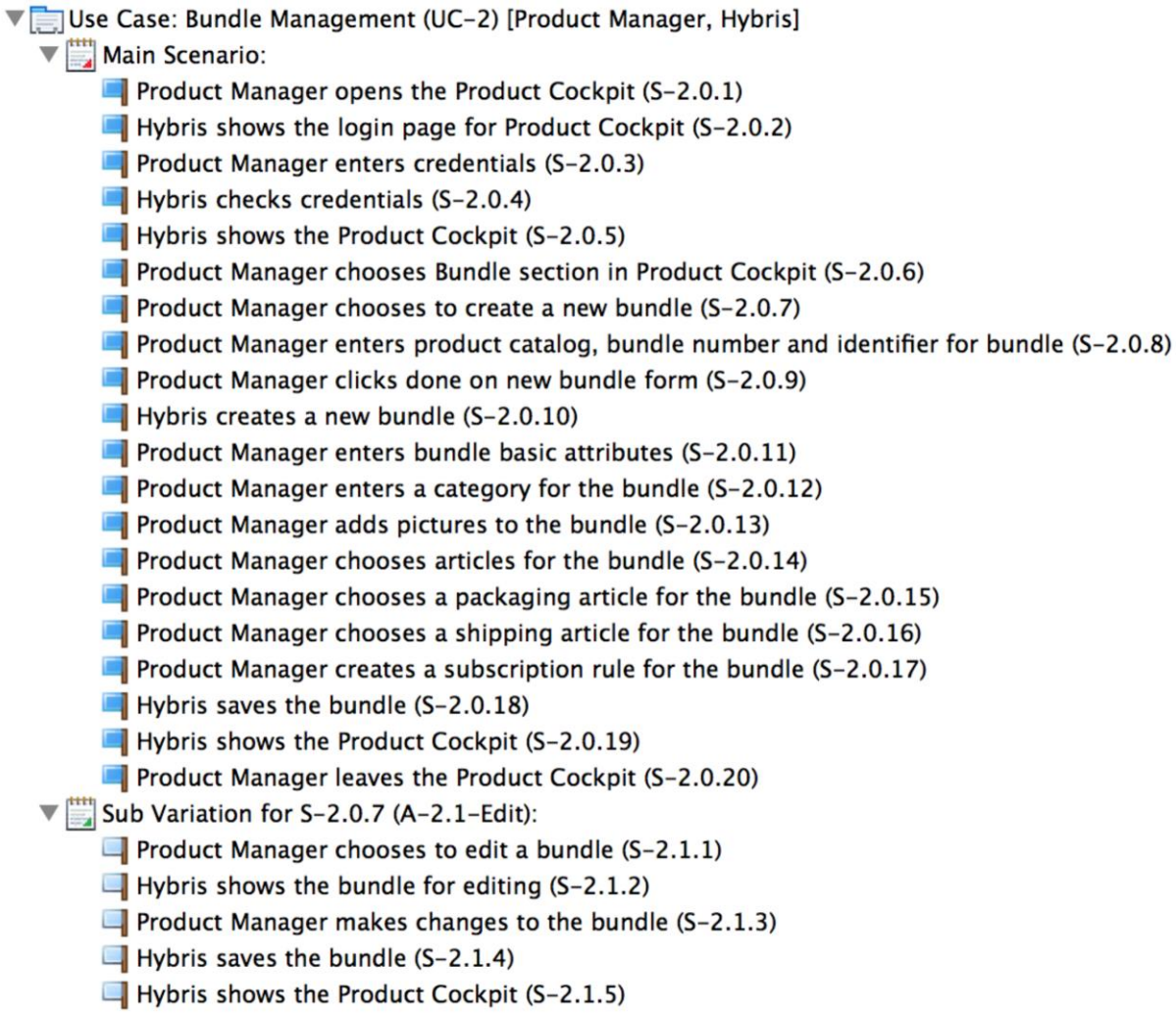


Figure 4.3. Bundle Management Use Case defined with IDM toolset

The actors for this Use Case are the Product Manager and Hybris. Product Manager is responsible for creating and maintaining the Bundles via the Hybris Product Cockpit. The main scenario deals with creating the bundle. There is an

alternative scenario (sub-variation “A-2.1-Edit”) representing the maintenance of an existing bundle. Accessing the Product Cockpit is the same as for the Article Management. The main attributes for a bundle are – articles (step “S-2.0.14”), shipping article (step “S-2.0.15”), packaging article (step “S-2.0.16”) and a subscription rule (step “S-2.0.17”). In Figure 4.3 a screenshot of the IDM Use Cases tool is shown with the Bundle Management Use Case “UC-2” expanded.

4.4.3. Product Management Use Case

Product Management is the final stage before the products can be published to the website and available for the customers.

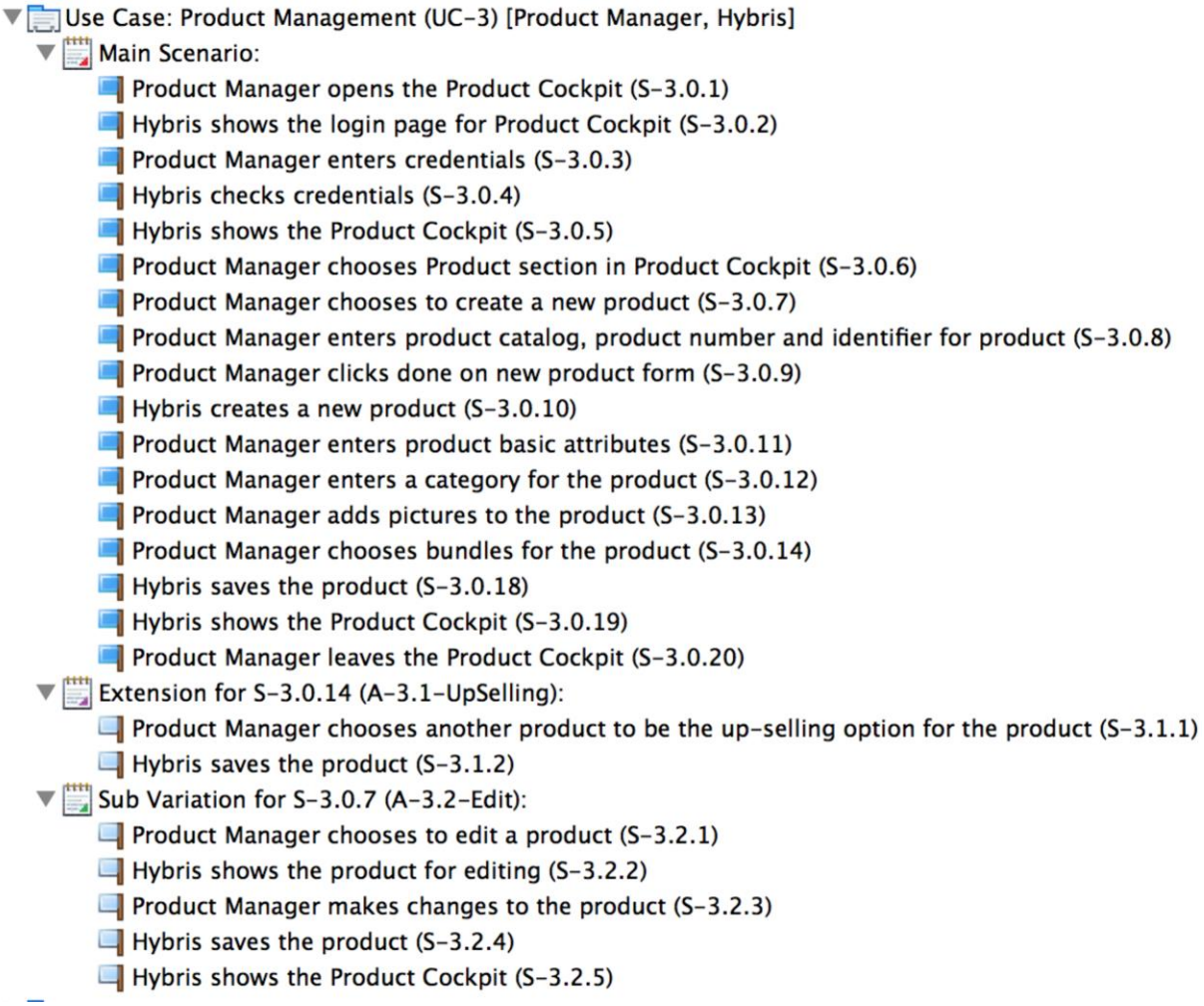


Figure 4.4. Product Management Use Case defined with IDM toolset

Product Manager and Hybris are the actors for this Use Case as well. Creation and maintenance covered by the main scenario and an alternative scenario (sub-

variation “A-3.2-Edit”) of the products is done via the Product Cockpit in the product section. Accessing the Product Cockpit is done in the same way as for Article and Bundle Management Use Cases. The main attribute for a product is the bundles (step “S-3.0.14”). There is an alternative scenario covering how up-selling options are added to the product (extension “A-3.1-UpSelling”). In Figure 4.4 a screenshot of the IDM Use Cases tool is shown with the Product Management Use Case “UC-3” expanded.

4.4.4. Customer Subscription Use Case

Customer Subscription process provides the possibility for the customers of SCB to subscribe to a particular product.

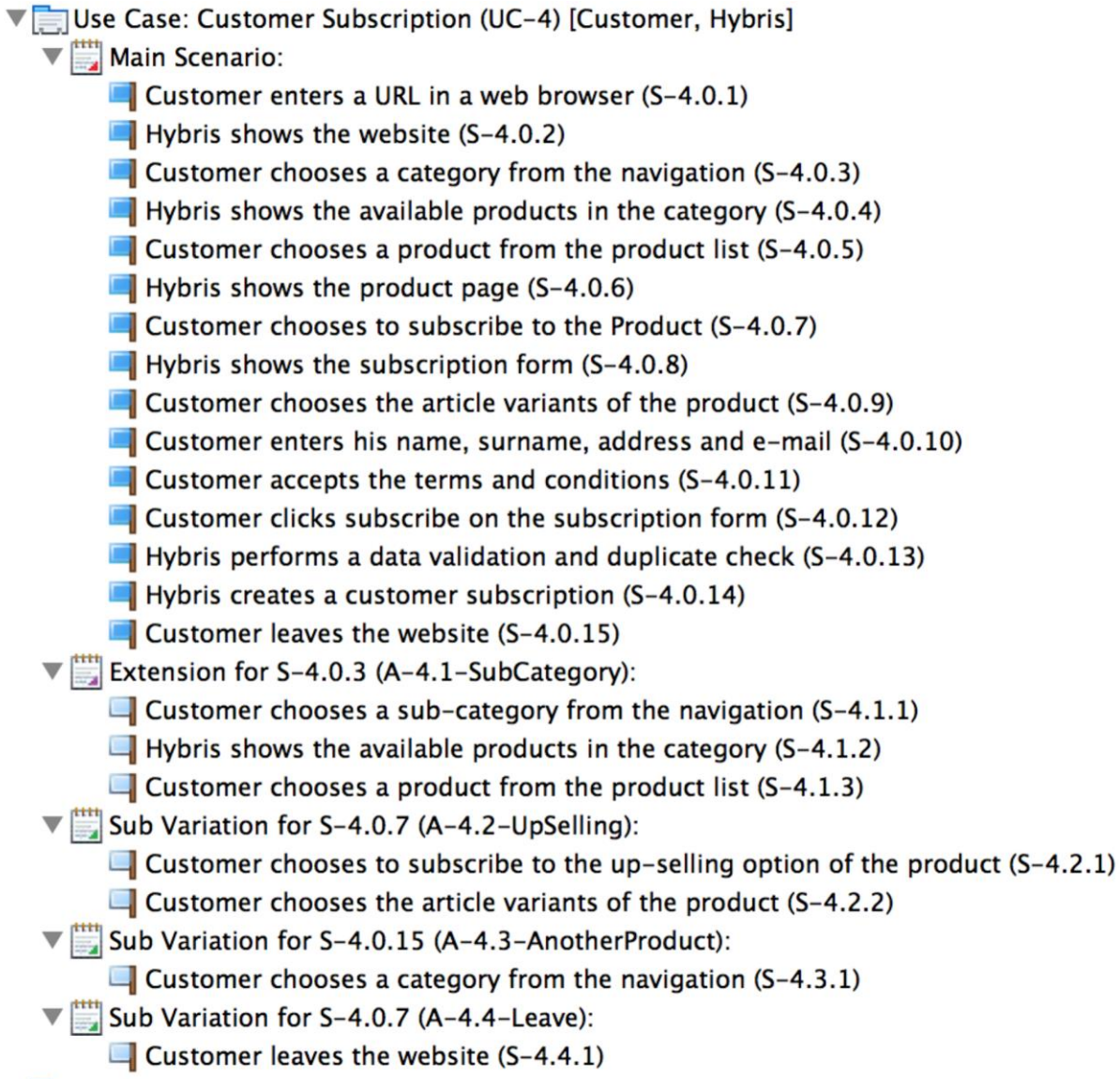


Figure 4.5. Customer Subscription Use Case defined with IDM toolset

The actors of this Use Case are Customer and Hybris (since Hybris also handles the SCB website). The subscription happens via the website; this is covered by the main scenario. The customer navigates through the categories (step "S-4.0.3") and chooses a product he wants to subscribe to (step "S-4.0.5"). The he must choose an article variant (step "S-4.0.9"), enter his customer details (step "S-4.0.10"), accept the terms and condition (step "S-4.0.11"), and approve the subscription (step "S-4.0.12"). In case the articles for this product do not have size, shape or style, then there will only be one article variant for the customer to choose. There are the following alternative scenarios – customer navigates to a sub-category instead of a first level category (extension "A-4.1-SubCategory"), customer chooses an up-selling option instead of the regular product (sub-variation "A-4.2-UpSelling"), customer wants to subscribe to another instead of leaving the website (sub-variation "A-4.3-AnotherProduct"), and customer doesn't want to subscribe to a product and leaves the website instead (sub-variation "A-4.4-Leave", which is an alternative to starting from step "S-4.0.7").

4.4.5. Bundle Delivery Use Case

Bundle Delivery process is part of ERP component of the system architecture. Here the author will cover it on a very high level with the goal of defining the main integration points between warehouse and e-commerce processes. This is also done because of the perspective of TFM to provide analysis of the main functional cycle [63] (this will be done in next section "Acquiring the TFM"). The actors for this Use Case are Warehouse Manager and ERP. The main scenario deals with updating the delivery schedule for next period (step "S-5.0.1") based on subscriptions (step "S-5.0.2"). Warehouse Manager needs to check the next shipment (step "S-5.0.4"), add marketing materials (step "S-5.0.5") and approve next shipment (step "S-5.0.7"). There is an alternative scenario (sub-variation "A-5.1-Override") when the Warehouse Manager has to override the generated shipment and do manual adjustments.

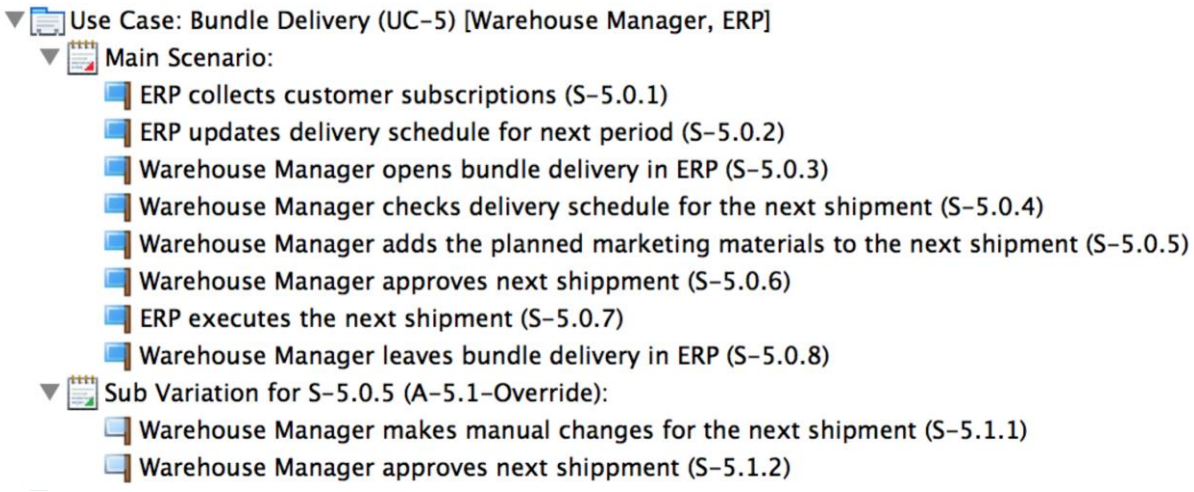


Figure 4.6. Bundle Delivery Use Case defined with IDM toolset

4.4.6. Batch Purchase Use Case

Batch Purchase process is part of ERP component of the system architecture. Same as with the Bundle Deliver, the author will cover it on a very high level with the goal of defining the main integration points between warehouse and e-commerce processes. The actors of this Use Case are Article Manager and ERP.

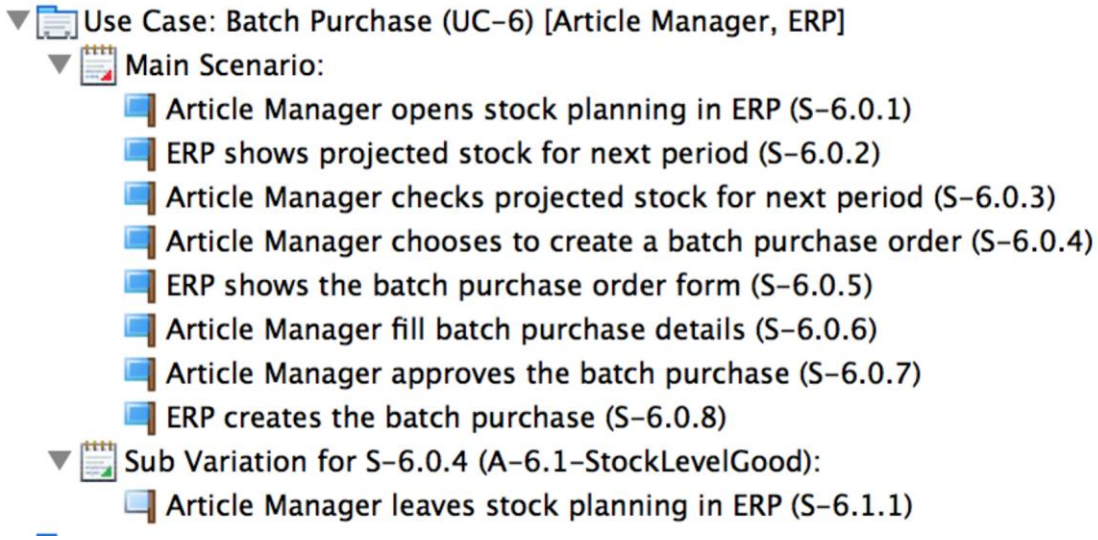


Figure 4.7. Batch Purchase Use Case defined with IDM toolset

The main scenario covers how the Article Manager monitors the stock plan (step "S-6.0.3") and creates batch purchase orders (step "S-6.0.7"). There is an alternative scenario (sub-variation "A-6.10StockLevelGood") after Article Manager

checks the projected stock (step “S-6.0.3”) and decides that the stock level is good and no batch purchase is necessary at this time.

4.5. Validating the Use Cases against Ontology

This sub-section shows the validation of the Use Cases against Ontology for the Product Subscription Business with IDM Use Cases Editor. Ontology can be used for validating the Use Cases as defined in the Section 2. Moreover, there are at least 2 problems that can be addressed – ambiguity and inconsistency. Ambiguity problem is when different terms are used to state the same thing, or a more general term is used to state something more specific. To address ambiguity of Use Case actors it is necessary to validate the actor of each step. First check is whether the noun phrase of the step matches with one of the actors defined for this Use Case. If not, then the whole list of actors needs to be checked. If even then there is no match, then the Ontology class hierarchy can be used for further checks. The noun phrase can be looked up in the Ontology class hierarchy and checked whether the class has a super-class or equivalent class that corresponds to the actors defined for the Use Case. In this case it is possible to trace the noun phrase to the corresponding actor, and the system analyst can then correct the ambiguity of the Use Case step. For example, the step “S-1.0.12” stating “Product Manager enters a category for article” can be considered. What if instead of “Product Manager” the system analyst has put “User“ as the noun phrase, i.e., “User enters a category for article”. This description also makes perfect sense, but is ambiguous since it is not clear which type of user it is. Since for the scope of Use Cases there are several types of users defined and used as actors, the system analyst needs to be more specific. As shown in Figure 4.1, class “User” is an equivalent class of “Person” class, which itself is a parent class of the following classes – “Customer”, “ArticleManager”, “ContentManager”, “ProductManager”, “CampaignManager” and “WarehouseManager”. Now it is possible to compare this list with the list of actors for the Use Case to find the applicable noun phrase for the Use Case step.

On the other hand, inconsistency is a problem when some verb phrase doesn’t make sense together with given noun phrase based on the business domain, i.e., Ontology. To address this inconsistency problem the Ontology object properties are

used. As shown in figure 3, object properties can be used in the class to define relationships with other classes. On the left hand side the class definition is shown and on the right hand side generated snipped in natural language are presented (ACE Snippets [24]). The following object properties relate to class “Hybris” – “check”, “create”, “save” and “show”. For each object property also the corresponding classes have been specified. For example, consider Use Case step “S-1.0.14” (as shown in Figure 4.2) stating “Hybris saves the article”. What if instead of “saves the article” as the verb phrase system analyst would put “stores the article”, so the description would be “Hybris stores the article”. Looking at the object properties of class “Hybris” there is no property for “store”, thus it is clear that the Use Case step is not valid. When doing a lookup of class “Article” in the object properties of class “Hybris” it is possible to see that there are to options for the system analyst to use – “save” or “create”.

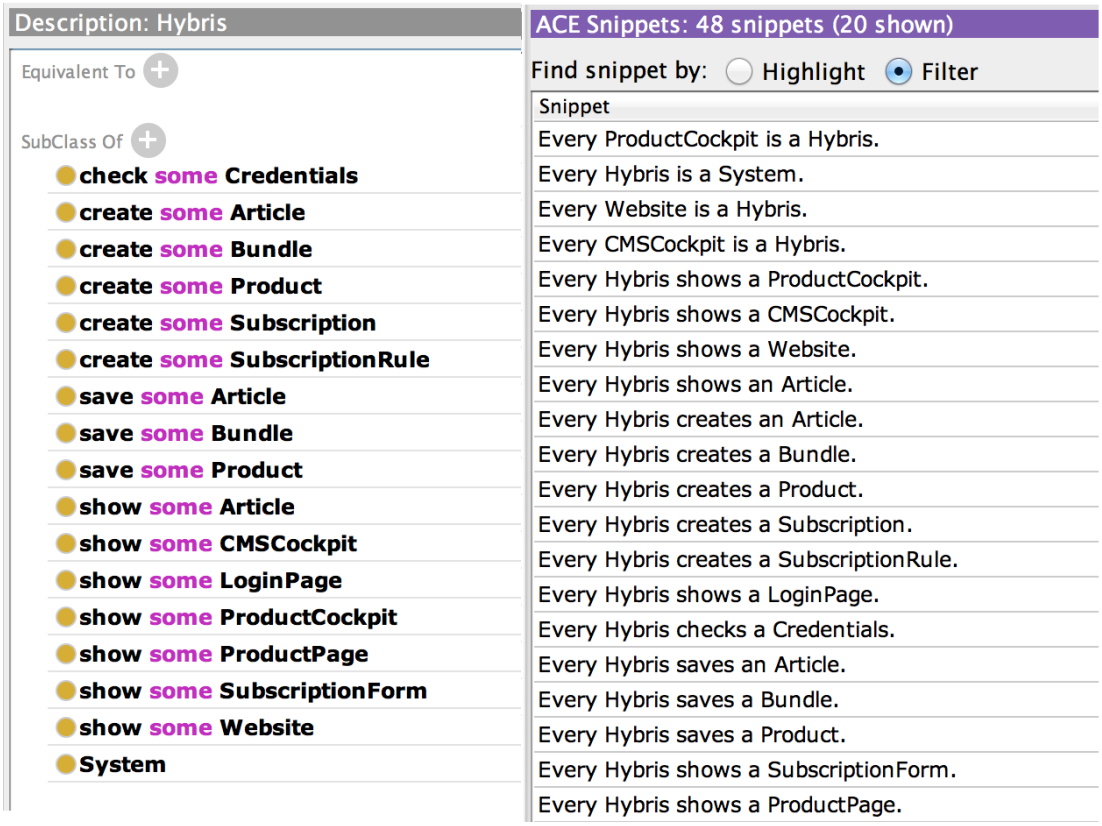


Figure 4.8. Text snippets based on Ontology

This is how Ontology can help validating the Use Cases. Some of these validation activities can be included as part of the IDM toolset and provide a level of automation, but currently this is done manually by the system analyst. For example,

the ambiguity problem of actors could be handled in the model-to-model transformation when Use Case step descriptions are parsed with a Statistical Parser to get the noun phrase and identify the entity for the TFM's functional feature. At that point it would be possible to validate the entity against the actor list and Ontology classes. However, this is currently a planned functionality of the IDM toolset, and is not implemented yet.

4.6. Acquiring the TFM

This sub-section shows the model-to-model transformation from Use Cases to TFM for the Product Subscription Business with IDM Use Cases to TFM Transformation tool. Use Cases and Ontology are important artifacts to describe the business domain – they provide the declarative and procedural knowledge. The next stage is to acquire a graphical representation of the business process, i.e., a domain model. IDM approach is based on the Topological Functioning Model (TFM), which is used for this purpose. Once the Use Cases are developed and have been validated, it is possible to do a model-to-model transformation using the IDM toolset's Use Cases to TFM Transformation tool. This automatically generates a TFM that corresponds to the defined Use Cases. The following sections show the TFM for the SCB process, which corresponds to the Use Cases defined in the previous sections. After acquiring the TFM the system analyst in addition to the generated topological relationships needs to link the processes together (which are based on the independent Use Cases) creating functioning cycles. System analyst also needs to identify the main functioning cycle.

4.6.1. Main Functioning Cycle

In the case of SCB the main functioning cycle goes through several processes – Batch Purchase, Customer Subscription and Bundle Delivery. These 3 processes define SCB's daily operation and enable the business. The articles need to be purchased, so that the customer can subscribe to products and get their bundles delivered. To link the 3 processes system analyst needs to create 3 topological relationships connecting the corresponding functional features. In addition to the main

functioning cycle there is also the cycle for creating and publishing the products in the first place, which covers the following processes – Article Management (shown in Figure 4), Bundle Management and Product Management. To link this cycle with the main functioning cycle the system analyst needs to create 2 additional topological relationships connecting Batch Purchase with Article Management and Product Management with Customer Subscription (the corresponding functional features). Thus just by adding 5 additional topological relationships the TFM is complete. In Figure 4.9 functional feature "S-1.0.1" has an incoming topological relationship. This is coming from the Batch Purchase process, since to be able to create an Article SCB would have already issued a batch purchase of the articles and would have the necessary initial data for the articles. As per the TFM after the login procedure the Product Manager is shown the Product Cockpit. From there he can choose the Article section (functional feature "S-1.0.6") or he can choose another section – Bundles or Product. Those are 2 other outgoing topological relationships from "S-1.0.15", and correspondingly there are also 2 incoming topological relationships after Bundle or Product is saved. The rest of the topological relationships of Article Management are shown in Figure 4.9. In the Article section of Product Cockpit the Product Manager can choose the operation to perform – create new article (starts with functional feature "S-1.0.7"), edit an article (starts with functional feature "S-1.1.1"), or create a new variant article (starts with functional feature "S-1.2.1").

4.6.2. Article Management in Topological Functioning Model

In the figure below the Article Management process is shown as part of the generated TFM. It reflects the same process as presented with the Use Cases (see Figure 4.2), but it is in a graph form and in addition it is possible to see the functioning cycles and interconnections with other processes. The incoming topological relationship on the right hand side comes from the Batch Purchase process. When the batch of articles has been bought, it has to be created in Hybris. The process starts with the Product Manager opening Product Cockpit and logging in. One of the functioning cycles for Article Management is the creation of articles. The cycle starts when Hybris shows the Product Cockpit (functional feature "S-1.0.15") and continues through the

creation of a new article (functional feature “S-1.0.7”) and ends with saving the article (functional feature “S-1.0.14”). This cycle can go on until the Product Manager has created all articles on his to-do list. Similarly, there is a functioning cycle for editing articles and creating article variants. There are 2 outgoing and 2 incoming topological relationships, which connect to Bundle Management and Product Management processes via functional feature “S-1.0.15”, since the process of logging in and showing the Product Cockpit is shared among these processes.

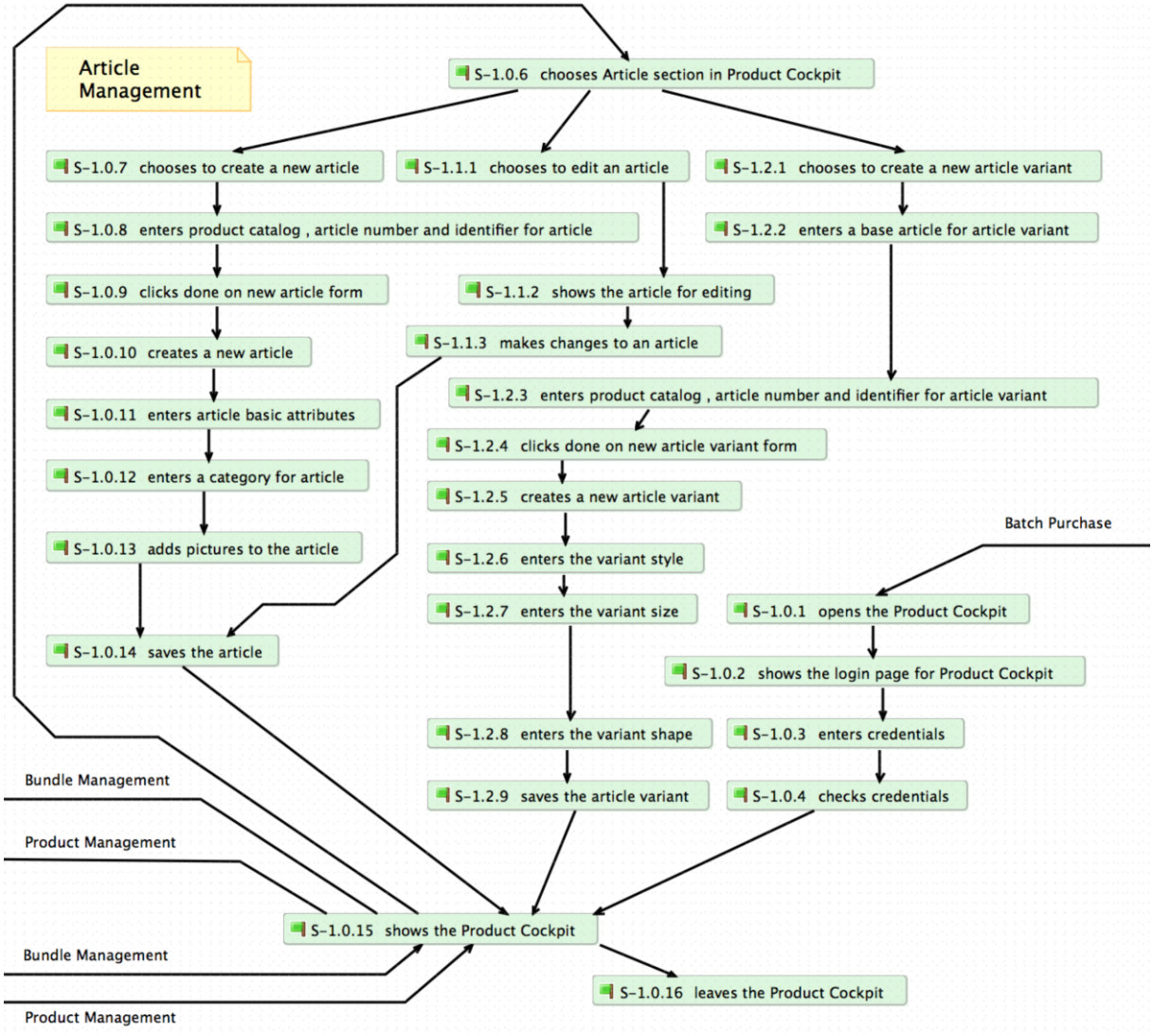


Figure 4.9. Article Management in form of a TFM

4.6.3. Bundle Management in Topological Functioning Model

In Figure 4.10 below the Bundles Management process is shown as part of the generated TFM.

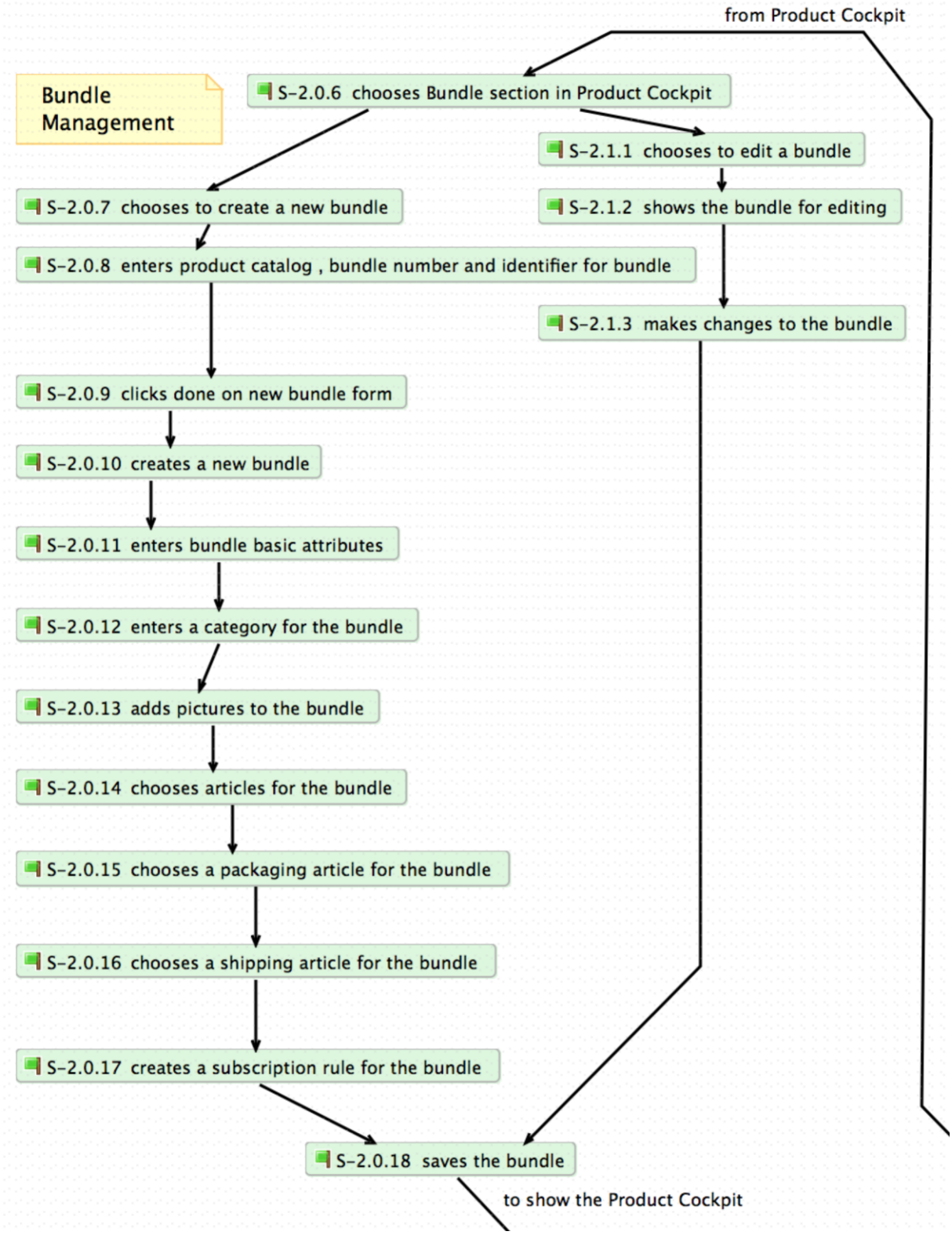


Figure 4.10. Bundle Management in form of a TFM

It reflects the same process as presented with the Use Cases (see Figure 4.3), presented in a graph form. Also in addition it is possible to see the functioning cycles and interconnections with other processes. The incoming topological relationship on the top right hand side comes from functional feature "S-1.0.15", which takes care of showing the Product Cockpit to the Product Manager. For the bundle management Product Manager chooses the Bundle section and then to add a new bundle. There is also a possibility to edit an existing bundle. After the bundle is saved Hybris will show the Product Cockpit again, which is the outgoing topological relationship from functional feature "S-2.0.18". Thus this provides at least 2 functioning cycles – creating a new bundle and editing an existing bundle.

4.6.4. Product Management in Topological Functioning Model

In Figure 4.11 below the Bundles Management process is shown as part of the generated TFM. It reflects the same process as presented with the Use Cases (see Figure 4.4), presented in a graph form. Also in addition it is possible to see the functioning cycles and interconnections with other processes. The incoming topological relationship on the top right hand side comes from functional feature "S-1.0.15", which takes care of showing the Product Cockpit to the Product Manager. For the product management Product Manager chooses the Product section and then to add a new product. There is also a possibility to edit an existing product. After the product is saved Hybris will show the Product Cockpit again, which is the outgoing topological relationship from functional feature "S-3.0.18". Thus this provides at least 2 functioning cycles – creating a new product and editing an existing product.

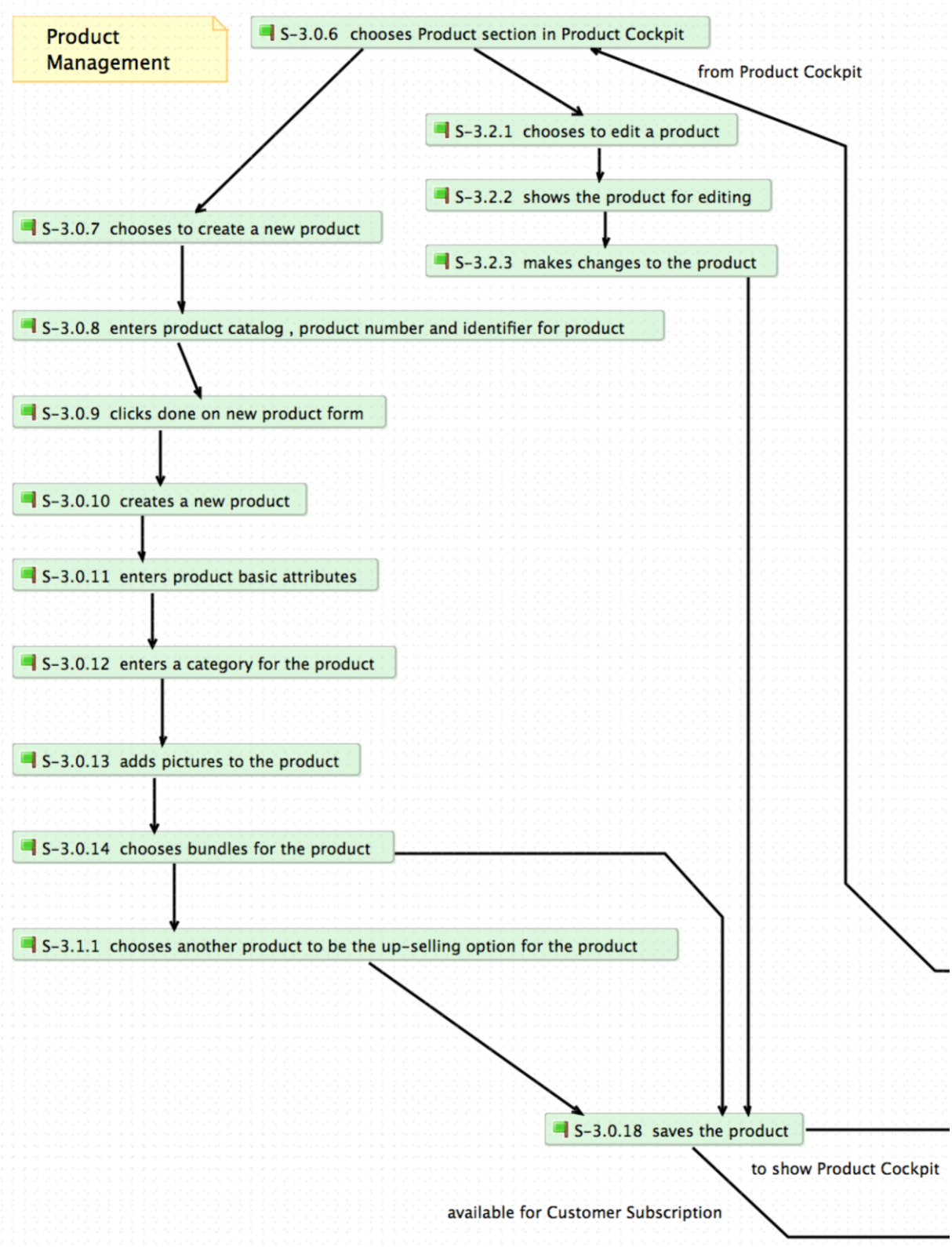


Figure 4.11. Product Management in form of a TFM

4.6.5. Customer Subscription in Topological Functioning Model

In Figure 4.12 below the Customer Subscription process is shown as part of the generated TFM. It reflects the same process as presented with the Use Cases (see Figure 4.5), presented in a graph form.

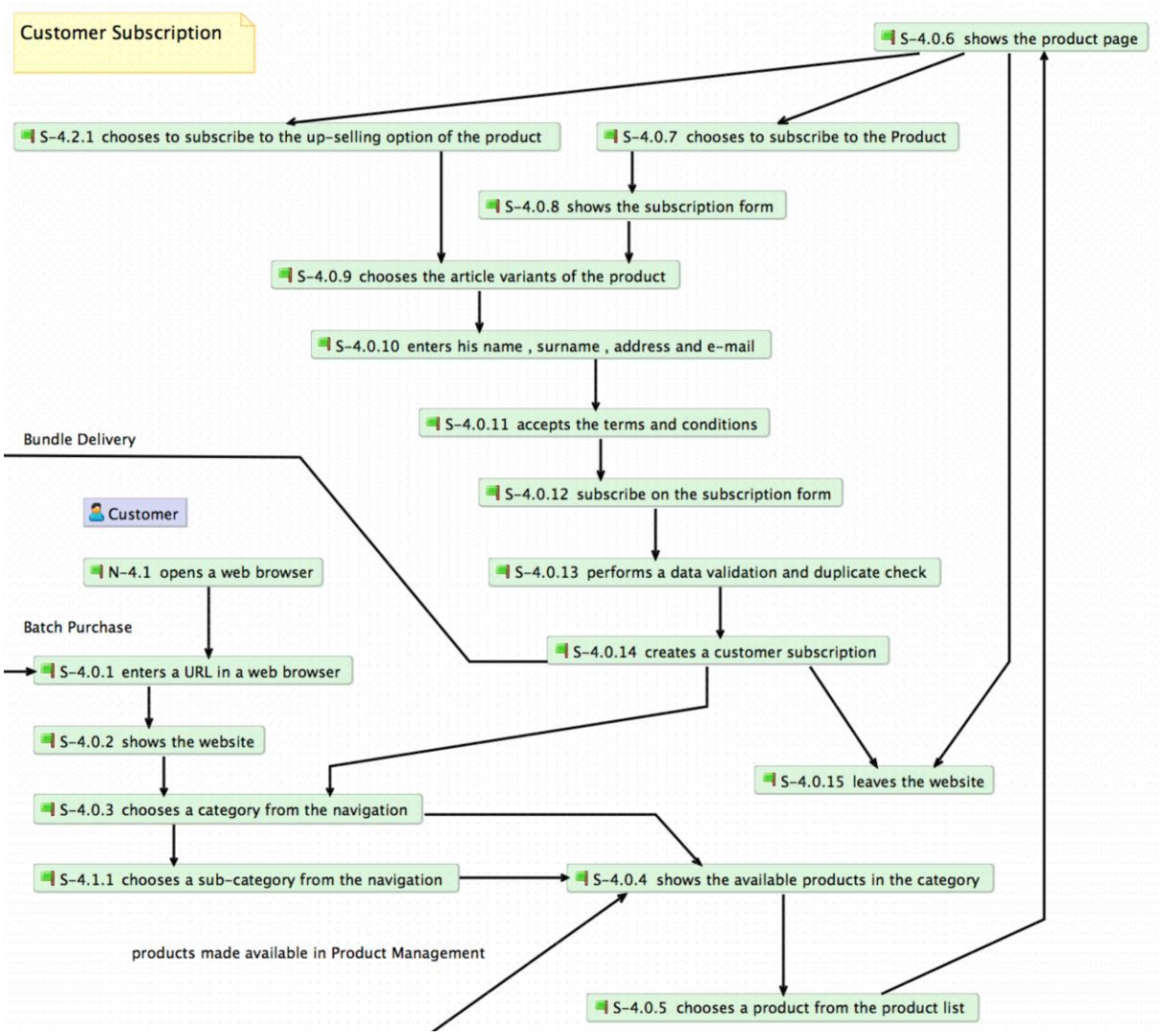


Figure 4.12. Customer Subscription in form of a TFM

Also in addition it is possible to see the functioning cycles and interconnections with other processes. There are 2 incoming topological relationships. The first is coming from the Batch Purchase process, so that the Customer is able to use the website and subscribe to products. And the second one is coming from the Product Management process, so that newly created product get published and shown to the Customer when he choose the corresponding category. The outgoing topological

relationship goes to the Bundle Delivery process, so that after the Customer has subscribed to a product he gets his deliveries. This is part of the main functioning cycle. There is a functioning cycle for subscribing to a product starting from the customer opening the web browser (functional feature "N-4.1") and ending with creating a customer subscription (functional feature "S-4.0.14"). "N-4.1" is an external functional feature it is added to the TFM manually by the system analysis. This functional feature is also one of the inputs of this TFM. One of the outputs of this TFM is when customer leaves the website after subscription or without subscription (functional feature "S-4.0.15").

4.6.6. Bundle Delivery in Topological Functioning Model

In Figure 4.13 below the Bundle Delivery process is shown as part of the generated TFM. It reflects the same process as presented with the Use Cases (see Figure 4.6), presented in a graph form. Also in addition it is possible to see the functioning cycles and interconnections with other processes. The Bundle Delivery process has an incoming topological relationship coming from Customer Subscription process to handle the subscriptions and ship the bundles to the customers. The outgoing topological relationship goes to the Batch Purchase process, so that it is possible to make sure there are enough articles in the warehouse to do the deliveries. This is part of the main functioning cycle. There is an external functional feature "N-5.1" created manually by the system analyst, which represents the handling of the shipment by the Postal Service. This is one of this TFM's outputs. As well as the functional feature "S-5.0.8", which represents the Warehouse Manager leaving the ERP system.

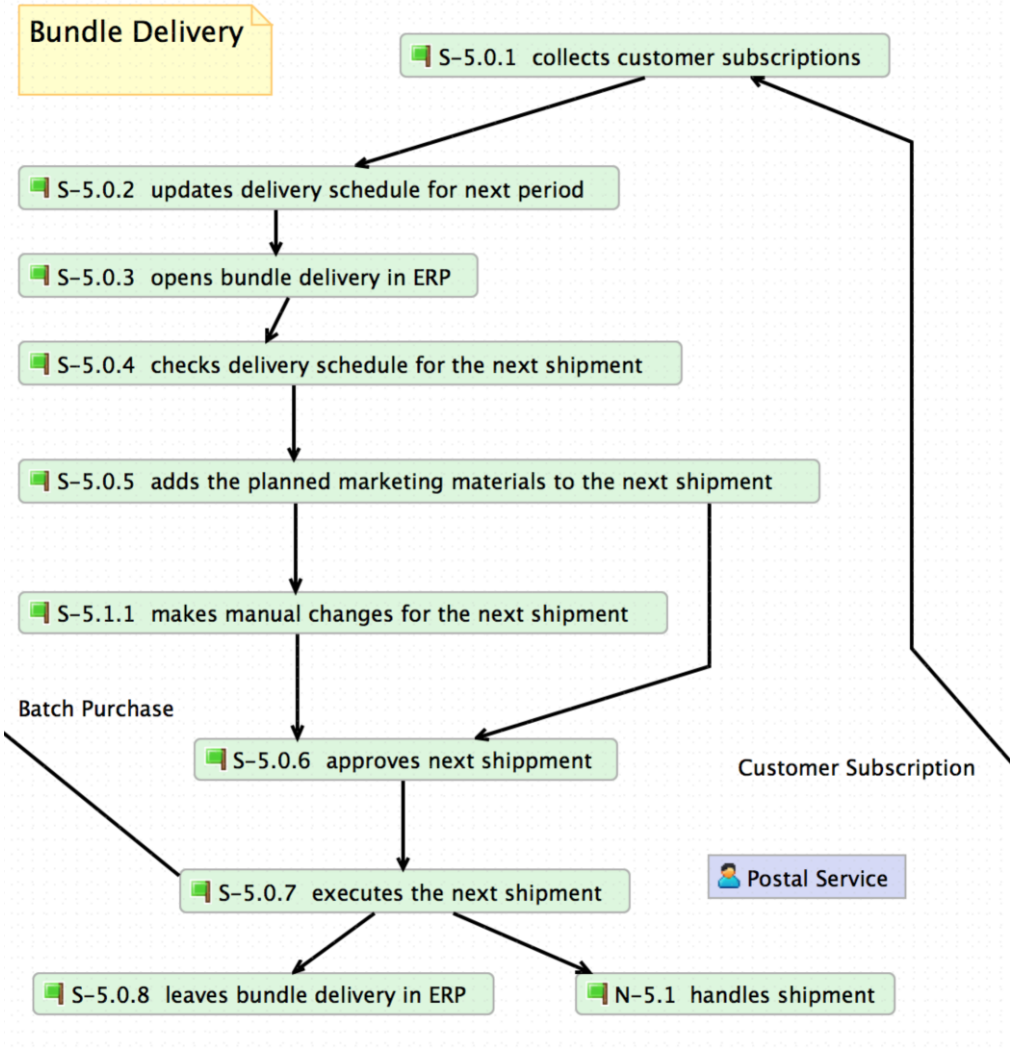


Figure 4.13. Bundle Delivery in form of a TFM

4.6.7. Batch Purchase in Topological Functioning Model

In Figure 4.14 below the Bundles Management process is shown as part of the generated TFM. It reflects the same process as presented with the Use Cases (see Figure 4.7), presented in a graph form. Also in addition it is possible to see the functioning cycles and interconnections with other processes. On the top right hand side there is an incoming topological relationship, which come from the Bundle Delivery process. This is so that the Article Manager can make sure there are enough articles in the warehouse to handle the planned deliveries. There are 2 outgoing topological relationships. The first one goes to Customer Subscription process when the batch purchase is created (functional feature “S-6.0.8”) and is part of the main

functioning cycle. After a batch process is created the stock level in the warehouse is enough to allow more customer subscriptions. The other outgoing topological relationship goes to the Article Management process when the batch purchase is created (functional feature "S-6.0.8") since sometimes new articles are purchased, which are not yet part of any product. Thus the Product Manager needs to create articles, bundles and a product for them. Functional feature "N-6.1" represents handling of the batch purchase order by the Supplier and has been added manually by the system analyst. This is an external functional feature and is one of the outputs of this TFM. Another output of this TFM is when the Article Manager leaves the stock planning in ERP (functional feature "S-6.1.1").

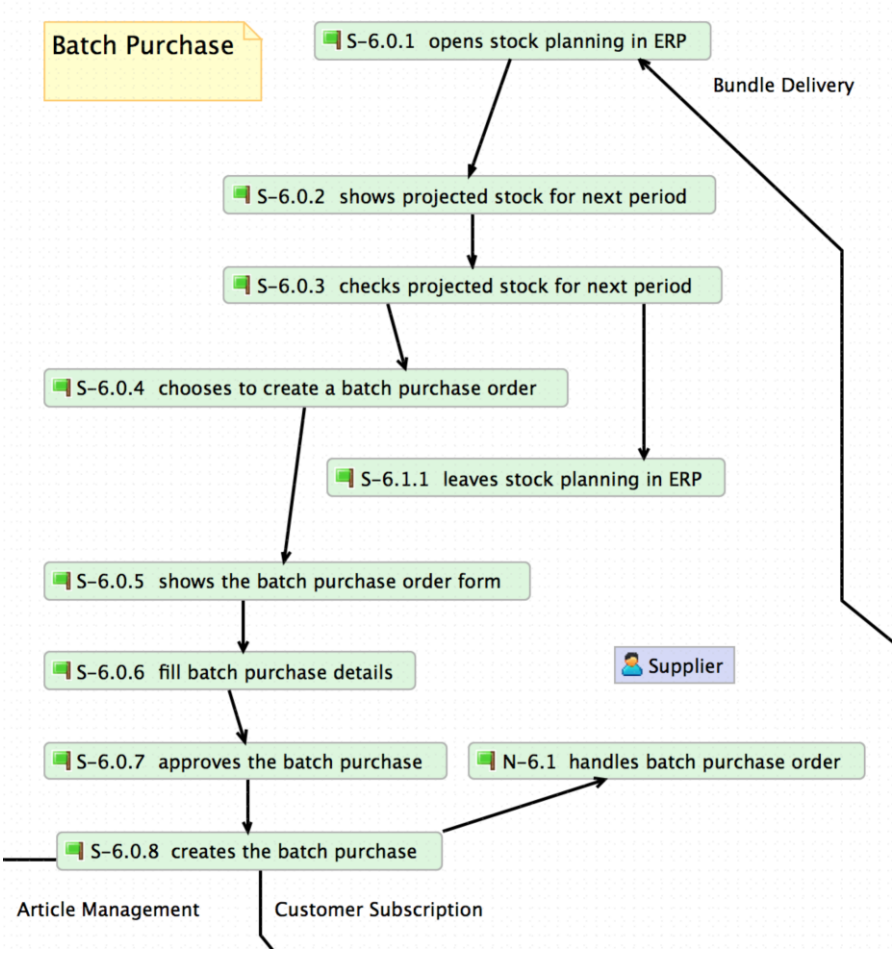


Figure 4.14. Batch Purchase in form of a TFM

4.7. Case Study Analysis

During the case study the following exercises were done: 1) initial analysis of the business domain; 2) defining the scope of the domain model; 3) building the Ontology for the business domain; 4) building the Use Cases for the business domain; 5) building the Topological Functioning Model (TFM) for the business domain. The Use Cases were built using the Integrated Domain Modeling (IDM) toolset developed by the author (details on the toolset are given in Section 3). The initial TFM for the business domain was acquired automatically using the IDM toolset based on the Use Cases and then manually corrected also using the IDM toolset.

The author will compare the IDM to TFM4MDA, which is the closest approach for domain modeling, since it also produces a TFM. The initial starting point of both approaches can be considered similar since there needs to be an initial introduction to the domain and definition of scope (both done together with the business people). Initially the scope boundaries are set on a process or component level, but later the scope is defined by inputs and outputs for both approaches, which are specifically identified at a later stage (might be in Use Cases, but must be in TFM). The first artifacts to be produced by each of the approaches are completely different. TFM4MDA expects an informal description (including and informal vocabulary for the description) in written natural language to be prepared. On the other hand, the IDM approach requires Use Cases to be produced together with the business people, and Ontology as the vocabulary for the domain. Based on the beginning of both approaches from training and effort perspective IDM is a bit heavier, since it requires production of Use Cases and Ontology as oppose to text. From perspective of understandability of the initial artifacts again IDM is a bit heavier for the business people to understand, since understanding Use Cases and Ontology is a pre-requisite (note that Use Cases are widely popular because of their simple structure, and Ontology mainly defines concepts and their relationships in a straightforward manner). However, from perspective of accuracy the IDM is superior, because the structure provided by the Use Cases and Ontology enable a better overview of the initial knowledge about the domain. From authors experience free text descriptions can be ambiguous and contain many errors, even if at the initial stage the business people

approve this description. A more structured approach makes the errors obvious and easier to correct. Nevertheless, at this initial stage the author considers IDM and TFM4MDA to be even (IDM a more complex, but a more precise and structured approach).

The real strength of IDM shows in the next stage – acquiring the TFM. To acquire the TFM with TFM4MDA a manual transformation from the informal description to TFM is required, which is a heavy, effort consuming process. First a list Functional Features are identified and then Topological Relationships between them are defined (based on the informal description). Moreover, this kind of manual labor causes human error based inconsistencies in the domain model. If there are errors, they can be detected in the representation of the TFM. Then it would be necessary to go back to the informal description, make corrections, and then make corrections also in the TFM (which again can be a heavy manual process). On the other hand, the IDM approach provides a model-to-model transformation that enables to automatically acquire the TFM from Use Cases. Thus, TFM is generated exactly corresponding to the previously defined knowledge about the domain. If any errors are found in the generated TFM, it is possible to go back to Use Cases, make correction and regenerate any number of times until the resulting model is approved. This introduces a significant level of automation for developing the domain model and saves effort. Because of this automation, acquiring a graphical representation of the business process is quick and easy after defining the Use Cases, so it is possible to use it in a trial and error fashion (if this is a personal preference, there are no penalties for this).

4.8. Summary

This case study was done based on a project done by Pearl Consulting AS for a customer who is in subscription commerce business (SCB), which operates in Norway, Sweden, Denmark, Finland and United Kingdom. The main business processes that were analyzed were Article Management, Bundle Management, Product Management, Customer Subscription, Bundle Delivery and Batch Purchase. Although Bundle Deliver and Batch Purchase were analyzed on a high level, since these processes are based on the ERP, but the main focus of the case study was E-commerce. The

Ontology, Use Cases and the TFM were part of the project's blueprint documentation delivered to SCB. It has been approved and the project is currently in its implementation phase.

The domain analysis and modeling was done according to the IDM approach. This provided a good structure to discuss and capture the business domain and model the TO-BE system. The standards used by the IDM approach – Ontology and Use Cases, were also very well known to the system analyst and business team. TFM provided a graphical overview of the process that could be acquiring with minimal effort using the IDM toolset. Moreover, TFM gives the opportunity for the system analyst to do functioning cycle analysis. This is a good way to trace and validate the business process. Overall, the IDM approach and toolset provided the necessary means for business domain analysis and modeling. The toolset provided an advantage for the projects since the artifacts produced during the analysis could be used as part of the blueprint.

CONCULSIONS

The goal of this doctoral thesis was to improve the process of domain analysis by providing an approach and a supporting toolset for acquiring a formal domain model that is transformable and can be used as CIM within MDA. The main result of this work is the development of the Integrated Domain Modeling (IDM) approach and toolset, which allows defining the business processes using Use Cases, validate them against corresponding Ontology and then generate a formal domain model automatically in the form of a Topological Functioning Model (TFM) with a model transformation. All of the tasks defined for achieving the goal have been successfully accomplished and the following results and conclusions have been obtained:

1. Results of analyzing the existing domain modeling approaches are as follows:
    a. Since domain model is intended to be used by the business people it should be simple to understand without special training, however this is not the case with existing approaches, which tend to be quite complex;
    b. Some of the approaches are strong with declarative knowledge and others are strong with procedural knowledge, however there is no approach to exploit both aspects of the domain to full extent;
    c. Use Cases is a simple approach for describing procedural knowledge and it is easy to understand by business people, however they are expresses in natural language and thus can be inconsistent, ambiguous, hard to manage, and hard to transform;
    d. Some novel approaches make use of Natural Language Processing (NLP) to deal with natural language texts for domain analysis, however they rely on an informal description in the beginning, which can be too unstructured and hard to manage to give enough input for the domain model;
    e. Tool support is very important for a domain modeling approach for the resulting domain model to be further used for software development;
    f. Computation Independent Model and Domain Model is the same thing in context of Model Drive Architecture, but not one and the same in context of domain modeling, since the scope is different;

2. Results of evaluating the domain modeling approaches based on their formality, conformance to MDA and practical usability:
    a. In comparison the domain modeling appproaches have a low level of formality providing only a meta-model, in contrast the TFM and Ontology provide a mathematically formal model and formal model validation, thus having a high score as formal approaches;
    b. BPMN, TFM and NIBA approaches are the most conformant to MDA from the analyzed approaches, since other approaches lack compatability to MOF and a model transformation to PIM/PSM;
    c. The highest score for practical usability was given to EPC for it also support declarative knowledge comparing to BPMN adn Use Cases;
    d. TFM approach scored low on practical usability because of lack of tool support, popularity and lack for support of decalarative knowledge;
    e. However in total based on all three sets of criteria TFM got the highest score mainly because of the formal domain model. Nonetheless, the weaknesses of TFM is practical usability, which can be improved and is addressed by the author;
3. Results of analyzing strengths, weaknesses and points for improvement of the TFM approach is as follows:
    a. The main strength of the TFM is its mathematical basis since it provides a formal way to capture the procedural aspects for the domain model;
    b. Another strength of the TFM is its inputs and outputs, which play an important role for identifying the scope of the domain model and enable the distinction between system and its environment;
    c. The third strength of the TFM is the cycle structure and the main functioning cycle, which provides the opportunity to validate the domain model;
    d. The forth strength of the TFM is the model-to-model transformations defined as part of TFM4MDA and improved with TopUML;
    e. One of the weaknesses of the TFM is that it describes the procedural knowledge for a domain and lacks the declarative knowledge;

f. Another weakness of the TFM and particularly TFM4MDA is that it depends on an informal description at its beginning, which is hard to manage and analyze;
g. The third weakness is that there is no tool support for TFM, TFM4MDA or TopUML, and as of now the only way to acquire a TFM is by heavy manual process;
h. Some suggested improvement for the TFM approaches include substitution of informal description with more formal domain knowledge, integration with declarative knowledge, model transformation within CIM and proper tool support.

4. The Integrated Domain Modeling (IDM) approach has been developed by the author for acquiring a formal domain model in the form of TFM based on formal knowledge about the domain using Use Cases and Ontolgoy as input. The IDM addresses the issues described above in the analysis;
5. A toolset to support the IDM approach has been developed by the author based on MDA standards and Eclipse EMF, GMF, M2M, which consists of Use Cases Editor, TFM Editor, and Use Cases to TFM Transformation tools. By exploiting the domain model acquired by the IDM toolset the system analyst together with the business can validate the business processes before the actual software developments start. Thus it is possible to make sure that the business processes correspond to the domain and also to the Ontology. This way debugging can be done in the domain model even before any line of code is written;
6. The case study for an e-commerce software development project demonstrated how the IDM approach and toolset can be successfully used to gather and record the domain knowledge, and acquire a domain model in a form of TFM via a model-to-model transformation; thus also acquiring a graphical representation of the business process automatically. Conclusions from conducting the case study are as follows:
    a. The IDM toolset requires minimal training (up to 1 day) to start using it, because the only mandatory pre-knowledge is Use Cases;
    b. IDM Use Cases are a very convenient way to document the business process during the blueprinting phase, because both the technical people as well as the business people understand it without special training;

c. A high level Ontology does help the domain analysis process, but going into details (e.g. object property definition) takes a lot of effort and is not mandatory (class hierarchy may well be enough as in case of the Subscription Commerce Business);
d. Acquiring a graphical representation of the business process is quick and easy if the Use Cases are defined (TFM is acquired automatically via model transformation, which corresponds to MDA standards);
e. Splitting the processes by Use Case and introducing Actor pools can improve graphical representation of the TFM.

Future research directions are as follows:

- In current state the IDM toolset's Use Case Editor is able to support the Use Case development process, but it is lacks the functionality to upload an Ontology and do automatic Use Case validation against the Ontology. This can be done by using Stanford Statistical Parser for Use Case analysis, OWL API for importing Ontology, and Eclipse EMF Validation Framework for validating the Use Cases model.
- Integration of the IDM toolset with UML tools by implementing the model transformations from TFM to UML described in TopUML research.

BIBLIOGRAPHY

[1] Alphabetical list of part-of-speech tags used in the Penn Treebank Project / Internet. - https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html [Accessed: June 14, 2014]

[2] ARIS / Internet. - http://www.ariscommunity.com/ [Accessed: June 14, 2014]

[3] Arlow J., Neustadt I. Introduction to BPMN 2. Mountain Way Limited, 2012. – 182 p.

[4] Asnina E. The Formal Approach to Problem Domain Modeling within Model Driven Architecture// In 9th International Conference on Information Systems Implementation and Modelling. - Prerov, Czech Republic, Ostrava, 2006. - pp. 97-104.

[5] Asnina E., Osis J. Topological Functioning Model as a CIM-Business Model// Model-Driven Domain Analysis and Software Development: Architectures and Functions. - IGI Global, 2011. - pp. 40-64.

[6] Baclawski K., Kokar M. K., Kogut P., Hart L., Smith J. E., Letkowski J., Emery P. Extending the Unified Modeling Language for Ontology Development// Int. Journal Software and Systems Modeling (SoSyM). - 2002. - Vol. 1, No. 2. - pp. 142-156.

[7] Broy M. Domain Modeling and Domain Engineering: Key Tasks in Requirements Engineering// Perspectives on the Future of Software Engineering. - Springer Berlin Heidelberg, 2013. - pp. 15-30.

[8] Business Process Model and Notation (BPMN) version 2.0.2 / Internet. - omg.org/spec/BPMN/2.0.2/PDF [Accessed: June 14, 2014]

[9] Caliusco M. L., Galli M. R., Ruidías H. J. Towards the integration of Ontologies in the context of MDA at CIM level// XVIII Congreso Argentino de Ciencias de la Computación. - 2012.

[10] Coleman D. A. Use Case Template: draft for discussion// Fusion Newsletter. - April 1998. Available. - http://www.engr.sjsu.edu/~fayad/current.courses/cmpe202-Fall2009/docs/lecture3/CmpE202-22-UC-Template-Ex-L3-3g.pdf

[11] Cranefield S. Networked knowledge representation and exchange using UML and RDF// Journal of Digital Information. - 2001. - Vol. 1, No. 8. Available. - https://journals.tdl.org/jodi/article/viewArticle/30/31 [Accessed: June 14, 2014]

[12] Donins U. Software Development with the Emphasis on Topology// In Proceeding of 13th East-European Conference on Advances in Databases and Information Systems (ADBIS 2009). - Volume 5968 of LNCS. Springer, 2010. pp. - 220-228.

[13] Doniņš U. Topological Unified Modeling Language: Development and Application. Doctoral thesis, Riga Technical University, 2012. - 224 p.

[14] Eclipse GMF: Graphical Modeling Framework / Internet. - http://wiki.eclipse.org/Graphical_Modeling_Framework [Accessed: June 14, 2014]

[15] Eclipse Papyrus / Internet. - http://www.eclipse.org/papyrus/ [Accessed: June 14, 2014]
[16] Eclipse Requirements Management Framework / Internet. - http://www.eclipse.org/rmf/ [Accessed: June 14, 2014]
[17] EMF Developer Guide: The Eclipse Modeling Framework (EMF) Overview. - 2005. / Internet. - http://help.eclipse.org/ganymede/index.jsp?topic=/org.eclipse.emf.doc/references/overview/EMF.html [Accessed: June 14, 2014]
[18] Evans E. Domain-driven Design: Tackling Complexity in the Heart of Software. - Addison-Wesley Professional, 2004. - 359 p.
[19] Falkovych K., Sabou M., Stuckenschmidt H. UML for the Semantic Web: Transfor-mation-Based Approaches// In Knowledge Transfor-mation for the Semantic Web. - IOS Press, 2003. - pp. 92–106. Available. - http://www.cwi.nl/~media/publications/UML_for_SW.pdf [Accessed: June 14, 2014]
[20] Fliedl G., Kop C., Mayr H. C., Salbrechter A., Vohringer J., Weber G., Winkler C. Deriving static and dynamic concepts from software requirements using sophisticated tagging// Data & Knowledge Engineering. - 2007. - Vol. 61, Iss. 3. - pp. 433-448.
[21] Fouad A. Embedding requirements within model-driven architecture// Software Quality Journal 19.2. - 2011. - pp. 411-430.
[22] Francu J., Hnetynka P. Automated Generation of Implementation from Textual System Requirements// In Proceedings of the 3rd IFIP TC 2 CEE-SET. - Brno, Czech Republic, Wroclawskiej. - 2008. pp. 15-28.
[23] Frankel D. S. Model Driven Architecture: Applying MDA to Enterprise Computing. Wiley, Indianapolis, 2003. - 352 p.
[24] Fuchs N. E., Kaljurand K., Kuhn T. Attempto Controlled English for Knowledge Repre-sentation// In Reasoning Web, Fourth International Summer School 2008, Lecture Notes in Computer Science 5224. - Springer, 2008. - pp. 104–124.
[25] Gasevic D., Djuric D., Devedzic V. Model Driven Architecture and Ontology Development. Springer, Heidelberg, 2006. - 311 p.
[26] Goodman N. D. Mathematics as an objective science// American mathematical monthly. - 1979. pp. 540-551.
[27] Google Scholar / Internet. - http://scholar.google.com [Accessed: June 14, 2014]
[28] Guarino N. Formal Ontology and Information Systems// In Proceedings of Formal Ontology and Information Systems. - Trento, Italy, IOS Press, Amsterdam, 1998. pp. 3–15.
[29] Jacobson I., Spence I. Use case 2.0: Scaling up, scaling out, scaling in for agile projects. Ivar Jacobson International, 2011. - 54 p.
[30] Jones C. Positive and negative innovations in software engineering. International Journal of Software Science and Computational Intelligence (IJSSCI) 1.2. - 2009. - pp. 20-30.
[31] Jurafsky D., Martin J. H. Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition. - Pearson Education, 2000. - 934 p.
[32] Kaindl H. Structural Requirements Language Definition, Defining the ReDSeeDS Languages / Internet. - http://publik.tuwien.ac.at/files/pub-et_13406.pdf

[33] Kardoš M., Drozdová M. Analytical method of CIM to PIM transformation in Model Driven Architecture (MDA)// Journal of Information and Organizational Sciences 34.1. - 2010. - pp. 89-99.
[34] Kaur A., Mansaf A. Role of Knowledge Engineering in the Development of a Hybrid Knowledge Based Medical Information System for Atrial Fibrillation// American Journal of Industrial and Business Management. Vol. 3., No 1. - 2013. - pp. 36-41.
[35] Kirikova M, Finke A., Grundspenkis J. What is CIM: an information system perspective// Advances in Databases and Information Systems. Vol. 5968. - Springer Berlin Heidelberg, 2010. - pp. 169-176.
[36] Kirikova M. Domain Modeling Approaches in IS Engineering. Model-Driven Domain Analysis and Software Development: Architectures and Functions. - IGI Global, 2011, pp. 388-406.
[37] Kolmogorov A. N., Fomin S. V. Introductory Real Analysis (Silverman, R. A., Ed.). - Mineola, NY: Courier Dover Publications, 1975. - 416 p.
[38] Kontio M. Architectural manifesto: Choosing MDA tools / Internet. - http://www.ibm.com/developerworks/library/wi-arch18.html [Accessed: June 14, 2014]
[39] Malan R., Bredemeyer D. Functional Requirements and Use Cases / Internet. - http://www.bredemeyer.com/pdf_files/functreq.pdf [Accessed: June 14, 2014]
[40] Mayr H. C., Kop C. A User Centered Approach to Requirements Modeling// In Modellierung. - 2002. - pp. 75-86.
[41] MDA: Model Driven Architecture / Internet. - http://www.omg.org/mda/ [Accessed: June 14, 2014]
[42] Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification / internet. - http://www.omg.org/cgi-bin/doc?ptc/07-07-07.pdf [Accessed: June 14, 2014]
[43] Miller J., Jishnu M. MDA Guide Version 1.0. 1. Object Management Group, 2003. - 51 p.
[44] Miller, Joaquin, and Jishnu Mukerji. "Model driven architecture (mda)." Object Management Group, Draft Specification ormsc/2001-07-01 (2001).
[45] Moore R., Lopes J. Paper templates// In TEMPLATE'06 1st International Conference on Template Production. - SciTePress, 1999.
[46] Murata T. Petri nets: Properties, analysis and applications// In Proceedings of the IEEE 77.4. - 1989. pp. 541-580.
[47] Nickols F. The Knowledge in Knowledge Management. The Knowledge Management Yearbook 2001–2002. - Butterworth-Heinemann, Boston, 2000. - pp. 12-21.
[48] Nikiforova, O., et al. "Development of the tool for transformation of the two-Hemisphere model to the UML class diagram: technical solutions and lessons learned." Proceedings of the 5th International Scientific Conference „Applied Information and Communication Technology. 2012.
[49] Noy N. F., McGuinness D. L. Ontology development 101: A guide to creating your first Ontology. Technical Report SMI-2001-0880, Stanford Medical Informatics. - 2001.
[50] OMG Meta Object Facility (MOF) Core Specification Version 2.4.2 OMG April 2014 / Internet. - http://www.omg.org/spec/MOF/ [Accessed: June 14, 2014

[51] OMG: Object Management Group / Internet. - omg.org [Accessed: June 14, 2014]
[52] Ondrej M, Richta K. The BPM to UML activity diagram transformation using XSLT// Dateso. - Vol. 9. - 2009. - pp. 119-129.
[53] Osis J. Investigating Troubles of Complex System Functioning and Category Theory // Cybernetic and Diagnosis, Volume. 4. - Riga: Zinatne, 1970. - pp. 15-20 (in Russian)"
[54] Osis J. Software Development with Topological Model in the Framework of MDA// In Proceedings of the 9th CAiSE International Workshop on Evaluation of Modeling Methods in Systems Analysis and Design (EMMSAD'2004) in connection with the CAiSE'2004. - Vol. 1. - RTU, Riga, 2004. pp. 211 – 220.
[55] Osis J., Asnina E. A Business Model to Make Software Development Less Intuitive// In Proceedings of the 2008 International Conference on Innovation in Software Engineering. - Vienna, Austria, IEEE Computer Society CPS, Los Alamitos, USA, 2008. - pp. 1240-1246.
[56] Osis J., Asnina E. Derivation of Use Cases from the Topological Computation Independent Business Model. Model-Driven Domain Analysis and Software Development: Architectures and Functions. - IGI Global, Hershey, New York, 2011. - pp. 65-89.
[57] Osis J., Asnina E. Enterprise Modeling for Information System Development within MDA// In 41th Annual Hawaii International Conference on System Sciences. - HICSS, USA, 2008. - pp. 490.
[58] Osis J., Asnina E. Is Modeling a Treatment for the Weakness of Software Engineering? Model-Driven Domain Analysis and Software Development: Architectures and Functions. - IGI Global, Hershey, New York, 2011. - pp. 1-14.
[59] Osis J., Asnina E. Model-Driven Domain Analysis and Software Development: Architectures and Functions. - IGI Global, Hershey, New York, 2011. - 487 p.
[60] Osis J., Asnina E. Topological Modeling for Model-Driven Domain Analysis and Software Development: Functions and Architectures. Model-Driven Domain Analysis and Software Development: Architectures and Functions. - IGI Global, Hershey, New York, 2011. - pp. 15-39.
[61] Osis J., Asnina E., Grave A. Formal Problem Domain Modeling within MDA// Communications in Computer and Information Science (CCIS). Software and Data Technologies. - Vol. 22. Springer-Verlag Berlin Heidelberg, 2008. - pp. 387-398.
[62] Osis J., Asnina E., Grave A. Computation Independent Modeling within the MDA// In Proceedings of the IEEE International Conference on Software Science, Technology and Engineering. - Herzlia, Israel, IEEE Computer Society, 2007. - pp. 22-34.
[63] Osis J., Asnina E., Grave A. Computation Independent Representation of the Problem Domain in MDA// J. Software Eng. - Vol. 2, iss. 1. - pp. 19--46. Available. - http://www.e-informatyka.pl/e-Informatica/Wiki.jsp?page=Volume2Issue1 [Accessed: June 14, 2014]
[64] Osis J., Asnina E., Grave A. Formal Computation Independent Model of the Problem Domain within the MDA. Information Systems and Formal Models// In Proceedings of the 10th International Conference ISIM'07. - Silesian University in Opava, Czech Republic, 2007. - pp. 47-54.

[65] Osis J., Asnina E., Grave A. MDA Oriented Computation Independent Modeling of the Problem Domain// In Proceedings of the 2nd International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2007). - Barcelona, Spain, 2007. - pp. 66 -71.
[66] Osis J., Donins U. Formalization of the UML Class Diagrams// Evaluation of Novel Approaches to Software Engineering. - Springer-Verlag, Berlin Heidelberg, New York, 2010. pp. 180-192.
[67] Ouvans C. From BPMN process models to BPEL web services// Web Services, 2006. ICWS'06. International Conference on. IEEE, 2006. - pp. 285-292.
[68] Overmyer S. Lavoie B. Rambow O. Conceptual Modeling through Linguistic Analysis Using LIDA// In Proceedings of 23rd International Conference on Software Engineering (ICSE 2001). - Toronto, Canada, 2001.
[69] Overmyer S., Lavoie B., Rambow O. Conceptual Modeling through Linguistic Analysis Using LIDA// In Proceedings of the 23rd International Conference on Software Engineering. - Toronto, Ontario, Canada, 2001. pp. 401-410.
[70] OWL: Web Ontology Language / Internet. - http://www.w3.org/TR/owl2-quick-reference/ [Accessed: June 14, 2014]
[71] Pearl Consulting / Internet. - http://www.pearlconsulting.no
[72] Plante F. Introducing the GMF Runtime, 2006. / Internet. - http://www.eclipse.org/articles/Article-Introducing-GMF/article.html [Accessed: June 14, 2014]
[73] Poesio M. Domain modelling and NLP: Formal Ontologies? Lexica? Or a bit of both?// Applied Ontology, Vol. 1, No. 1. IOS Press, 2005. - pp. 27–33.
[74] Prieto-Díaz R. Domain Analysis: An Introduction// ACM SIGSOFT Software Engineering Notes 15.2. - 1990. - pp. 47-54.
[75] Protege / Internet. - http://protege.stanford.edu [Accessed: June 14, 2014]
[76] Santorini B. Part-Of-Speech Tagging Guidelines for the Penn Treebank Project// Technical report MS-CIS-90-47, Department of Computer and Information Science, University of Penllsylvania, 1990.
[77] SAP AG / Internet. - http://www.sap.com
[78] Scheer A. W., Oliver T., Otmar A. Process modeling using event-driven process chains// Process-Aware Information Systems. - 2005. - pp. 119-146.
[79] Šlihte A. Implementing a Topological Functioning Model Tool// In Scientific Journal of Riga Technical University, 5. series., Computer Science. - Vol. 43. - Riga, 2010. - pp. 68–75.
[80] Šlihte A. Introduction to Integrated Domain Modeling Toolset // Scientific Journal of RTU. Computer Science. - 2014. (to be published)
[81] Šlihte A. The Concept of a Topological Functioning Model Construction Tool// In 13th East-European Conference, ADBIS 2009, Associated Workshops and Doctoral Consortium, Local Proceedings. JUMI, Riga, Latvia, 2009. - pp. 476-484.
[82] Šlihte A. The Specific Text Analysis Tasks at the Beginning of MDA Life Cycle// In Data-bases and Information Systems Doctoral Consortium. Latvia, Riga, July 5-7, 2010. - pp. 11–22.
[83] Šlihte A. Transforming Textual Use Cases to a Computation Independent Model// MDA & MTDD 2010. Greece, Athens, July 22-24, 2010. - pp. 33–42.

[84] Šlihte A. Using Use Cases for Domain Modeling// In Proceedings of the 7th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2012) - 2012. - pp. 224-231.
[85] Šlihte A., Osis J., Doniņš U. Knowledge Integration for Domain Modeling// In Proceedings of the 3rd International Workshop on Model-Driven Architecture and Modeling-Driven Software Development. China, Bejing, June 8-11, 2011. - pp. 46-56.
[86] Soley R. Model driven architecture. OMG white paper, 2000. Available. - http://www.geocities.ws/pravin_suman/Resources/00-11-05.pdf [Accessed: June 14, 2014]
[87] Subramaniam K., Liu D., Far B., Eberlein A. UCDA: Use Case Driven Development Assistant Tool for Class Model Generation// In Proceedings of the 16th SEKE. Banff, Canada, 2004. Available. - http://enel.ucalgary.ca/People/eberlein/publications/SEKE-Kalaivani.pdf [Accessed: June 14, 2014]
[88] The Stanford Parser: A statistical parser. The Stanford Natural Language Processing Group, 2010 / Internet. - http://nlp.stanford.edu/software/lex-parser.shtml [Accessed: June 14, 2014]
[89] Van Lamsweerde A. Requirements engineering: from craft to discipline// In Proceedings of the 13th international Workshop on Early Aspects. - New York: Association for Computing Machinery, Inc, 2008. - pp. 238-249.
[90] W3C, OWL Web Ontology Language Overview, W3C Recommendation February 10 2004 / Internet. - http://www.w3.org/TR/owl-features/ [Accessed: June 14, 2014]
[91] White S. A., Introduction to BPMN. IBM Cooperation, 2004. Available. - http://yoann.nogues.free.fr/IMG/pdf/07-04_WP_Intro_to_BPMN_-_White-2.pdf [Accessed: June 14, 2014]
[92] XML Metadata Interchange (XMI) Specification, v2. 3.2, 2014 /Internet. - http://www.omg.org/spec/XMI/ [Accessed: June 14, 2014]
[93] Tsai A., Wang J., Tepfenhart W., Rosca, D: EPC workflow model to WIFA model conversion// In Proceedings of Systems, Man and Cybernetics, 2006. SMC'06. IEEE International Conference, Vol. 4, pp. 2758-2763
[94] Suchanek F.M., Kasneci G., Weikum G. Yago: A large Ontology from wikipedia and wordnet// Web Semantics: Science, Services and Agents on the World Wide Web 6.3, 2008, pp. 203-217.
[95] Ferrario R., Oltramari A.: A first-order cutting process ontology for sheet metal parts// Formal Ontologies Meet Industry 198, 2009, pp. 22.
[96] Jones C.: Software project management practices: Failure versus success// CrossTalk: The Journal of Defense Software Engineering 17, 2004.
[97] Kruczynski K.: Business process modelling in the context of SOA–an empirical study of the acceptance between EPC and BPMN// World Review of Science, Technology and Sustainable Development 7.1, 2010, pp. 161-168.
[98] Nüttgens M., Feld T., Zimmermann V.: Business Process Modeling with EPC and UML: transformation or integration?// The Unified Modeling Language. Physica-Verlag HD, 1998, pp. 250-261.
[99] Strommer M, Murzek M., Wimmer M.: Applying model transformation by-example on business process modeling languages// Advances in Conceptual

Modeling–Foundations and Applications, Springer Berlin Heidelberg, 2007, pp. 116-125.
[100]Bodrow W.: The dynamic of professional knowledge utilized in software applications for process controlling// Advances in Manufacturing, 2014, pp. 1-6.

APPENDICES

LIST OF FIGURES

4.1	4	Ontology for SBC
4.2	4	Article Management Use Case defined with IDM toolset
4.3	4	Bundle Management Use Case defined with IDM toolset
4.4	4	Product Management Use Case defined with IDM toolset
4.5	4	Customer Subscription Use Case defined with IDM toolset
4.6	4	Bundle Delivery Use Case defined with IDM toolset
4.7	4	Batch Purchase Use Case defined with IDM toolset
4.8	4	Text snippets based on Ontology
4.9	4	Article Management in form of a TFM
4.10	4	Bundle Management in form of a TFM
4.11	4	Product Management in form of a TFM
4.12	4	Customer Subscription in form of a TFM
4.13	4	Bundle Delivery in form of a TFM
4.14	4	Batch Purchase in form of a TFM

LIST OF TABLES

IDM USE CASES METAMODEL ACCORDING TO ECORE

```
<?xml version="1.0" encoding="UTF-8"?>
<ecore:EPackage xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore" name="usecases" nsURI="http://lv/rtu/ldk/idm/usecases/1.0" nsPrefix="usecases">
  <eClassifiers xsi:type="ecore:EClass" name="UseCases">
    <eStructuralFeatures xsi:type="ecore:EReference" name="actors" lowerBound="1"
        eType="#//Actors" containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="conditions" eType="#//Conditions"
        containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="useCase" lowerBound="1"
        upperBound="-1" eType="#//UseCase" containment="true" eOpposite="#//UseCase/useCases"
        eKeys="#//UseCase/id"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="domain" lowerBound="1"
        eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="scope" lowerBound="1" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="Ontology" lowerBound="1"
        eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="UseCase">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="id" lowerBound="1" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"
        iD="true"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="description" lowerBound="1"
        eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="actors" lowerBound="1"
        upperBound="-1" eType="#//Actor"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="mainScenario" lowerBound="1"
        eType="#//MainScenario" containment="true" eOpposite="#//MainScenario/mainScenarioUseCase"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="extensions" upperBound="-1"
        eType="#//Extension" containment="true" eOpposite="#//Extension/extensionUseCase"
        eKeys="#//AlternativeScenario/id"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="subVariations" upperBound="-1"
        eType="#//SubVariation" containment="true" eOpposite="#//SubVariation/subVariationUseCase"
        eKeys="#//AlternativeScenario/id"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="useCases" lowerBound="1"
        eType="#//UseCases" eOpposite="#//UseCases/useCase"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="SingleEvent" abstract="true" eSuperTypes="#//Event">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="description" lowerBound="1"
        eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="triggers" upperBound="-1"
```

```
40.         eType="#//Event"/>
41.     <eStructuralFeatures xsi:type="ecore:EAttribute" name="reference" eType="ecore:E
    DataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
42.   </eClassifiers>
43.   <eClassifiers xsi:type="ecore:EClass" name="MainScenario" eSuperTypes="#//Scenario
    ">
44.     <eStructuralFeatures xsi:type="ecore:EReference" name="step" lowerBound="1" uppe
    rBound="-1"
45.         eType="#//DefaultEvent" containment="true" eKeys="#//Event/id"/>
46.     <eStructuralFeatures xsi:type="ecore:EReference" name="mainScenarioUseCase" lowe
    rBound="1"
47.         eType="#//UseCase" eOpposite="#//UseCase/mainScenario"/>
48.   </eClassifiers>
49.   <eClassifiers xsi:type="ecore:EClass" name="Extension" eSuperTypes="#//Alternative
    Scenario">
50.     <eStructuralFeatures xsi:type="ecore:EReference" name="extensionUseCase" lowerBo
    und="1"
51.         eType="#//UseCase" eOpposite="#//UseCase/extensions"/>
52.   </eClassifiers>
53.   <eClassifiers xsi:type="ecore:EClass" name="SubVariation" eSuperTypes="#//Alternat
    iveScenario">
54.     <eStructuralFeatures xsi:type="ecore:EReference" name="subVariationUseCase" lowe
    rBound="1"
55.         eType="#//UseCase" eOpposite="#//UseCase/subVariations"/>
56.   </eClassifiers>
57.   <eClassifiers xsi:type="ecore:EClass" name="DefaultEvent" eSuperTypes="#//SingleEv
    ent"/>
58.   <eClassifiers xsi:type="ecore:EClass" name="AlternativeScenario" abstract="true"
59.       eSuperTypes="#//Scenario">
60.     <eStructuralFeatures xsi:type="ecore:EReference" name="reference" lowerBound="1"

61.         eType="#//DefaultEvent" eKeys="#//Event/id"/>
62.     <eStructuralFeatures xsi:type="ecore:EAttribute" name="id" lowerBound="1" eType=
    "ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"
63.         iD="true"/>
64.     <eStructuralFeatures xsi:type="ecore:EReference" name="step" lowerBound="1" uppe
    rBound="-1"
65.         eType="#//AlternativeEvent" containment="true" eKeys="#//Event/id"/>
66.   </eClassifiers>
67.   <eClassifiers xsi:type="ecore:EClass" name="Actor">
68.     <eStructuralFeatures xsi:type="ecore:EAttribute" name="description" eType="ecore
    :EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
69.   </eClassifiers>
70.   <eClassifiers xsi:type="ecore:EClass" name="Scenario" abstract="true"/>
71.   <eClassifiers xsi:type="ecore:EClass" name="AlternativeEvent" eSuperTypes="#//Sing
    leEvent"/>
72.   <eClassifiers xsi:type="ecore:EClass" name="Actors">
73.     <eStructuralFeatures xsi:type="ecore:EReference" name="actor" lowerBound="1" upp
    erBound="-1"
74.         eType="#//Actor" containment="true"/>
75.   </eClassifiers>
76.   <eClassifiers xsi:type="ecore:EClass" name="Condition" abstract="true">
77.     <eStructuralFeatures xsi:type="ecore:EAttribute" name="id" lowerBound="1" eType=
    "ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EInt"/>
78.   </eClassifiers>
79.   <eClassifiers xsi:type="ecore:EClass" name="Conditions">
80.     <eStructuralFeatures xsi:type="ecore:EReference" name="condition" upperBound="-
    1"
81.         eType="#//SingleCondition" containment="true"/>
82.     <eStructuralFeatures xsi:type="ecore:EReference" name="compositeCondition" upper
    Bound="-1"
83.         eType="#//CompositeCondition" containment="true"/>
84.   </eClassifiers>
85.   <eClassifiers xsi:type="ecore:EClass" name="CompositeCondition" eSuperTypes="#//Co
    ndition">
86.     <eStructuralFeatures xsi:type="ecore:EReference" name="conditions" lowerBound="2
    "
87.         upperBound="-1" eType="#//Condition"/>
```

```
88.     <eStructuralFeatures xsi:type="ecore:EAttribute" name="operation" eType="#//Oper
    ation"/>
89.   </eClassifiers>
90.   <eClassifiers xsi:type="ecore:EEnum" name="Operation">
91.     <eLiterals name="AND"/>
92.     <eLiterals name="OR" value="1"/>
93.     <eLiterals name="XOR" value="2"/>
94.   </eClassifiers>
95.   <eClassifiers xsi:type="ecore:EClass" name="Event" abstract="true">
96.     <eStructuralFeatures xsi:type="ecore:EAttribute" name="id" lowerBound="1" eType=
    "ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"
97.         iD="true"/>
98.     <eStructuralFeatures xsi:type="ecore:EReference" name="preconditions" eType="#//
    Condition"
99.         eKeys="#//Condition/id"/>
100.          <eStructuralFeatures xsi:type="ecore:EReference" name="postconditions" eT
    ype="#//Condition"
101.              eKeys="#//Condition/id"/>
102.        </eClassifiers>
103.        <eClassifiers xsi:type="ecore:EClass" name="SingleCondition" eSuperTypes="#
    //Condition">
104.          <eStructuralFeatures xsi:type="ecore:EAttribute" name="description" lower
    Bound="1"
105.              eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EStrin
    g"/>
106.        </eClassifiers>
107.      </ecore:EPackage>
```

IDM TFM METAMODEL ACCORDING TO ECORE

```
<?xml version="1.0" encoding="UTF-8"?>
<ecore:EPackage xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore" name="tfm" nsURI="http://ldi.rtu.lv/tfm/1.0" nsPrefix="tfm">
  <eClassifiers xsi:type="ecore:EClass" name="TFM">
    <eStructuralFeatures xsi:type="ecore:EReference" name="functionalFeatures" lowerBound="2"
        upperBound="-1" eType="#//FunctionalFeature" containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="topologicalRelationships"
        lowerBound="2" upperBound="-1" eType="#//TopologicalRelationship" containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="logicalRelationships" upperBound="-1"
        eType="#//LogicalRelationship" containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="actors" upperBound="-1"
        eType="#//Actor" containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="cycles" lowerBound="1"
        upperBound="-1" eType="#//Cycle" containment="true"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="FunctionalFeature">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="id" lowerBound="1" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"
        iD="true"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="description" lowerBound="1"
        eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="action" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="result" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="object" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="entity" eType="#//Actor"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="precond" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="postcond" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="subordination" eType="#//Subordination"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="executorIsSystem" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EBoolean"
        defaultValueLiteral="false"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Cycle">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="order" lowerBound="1" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EInt"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="isMain" lowerBound="1"
        eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EBoolean"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="functionalFeatures" lowerBound="2"
        upperBound="-1" eType="#//FunctionalFeature"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Actor">
```

```
39.     <eStructuralFeatures xsi:type="ecore:EAttribute" name="description" lowerBound="
    1"
40.         eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
41.   </eClassifiers>
42.   <eClassifiers xsi:type="ecore:EClass" name="TopologicalRelationship">
43.     <eStructuralFeatures xsi:type="ecore:EAttribute" name="id" lowerBound="1" eType=
    "ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
44.     <eStructuralFeatures xsi:type="ecore:EReference" name="source" lowerBound="1"
45.         eType="#//FunctionalFeature"/>
46.     <eStructuralFeatures xsi:type="ecore:EReference" name="target" lowerBound="1"
47.         eType="#//FunctionalFeature"/>
48.   </eClassifiers>
49.   <eClassifiers xsi:type="ecore:EClass" name="LogicalRelationship">
50.     <eStructuralFeatures xsi:type="ecore:EAttribute" name="id" eType="ecore:EDataTyp
    e http://www.eclipse.org/emf/2002/Ecore#//EString"
51.         iD="true"/>
52.     <eStructuralFeatures xsi:type="ecore:EAttribute" name="operation" eType="#//Logi
    calOperation"/>
53.     <eStructuralFeatures xsi:type="ecore:EReference" name="relatedElements" lowerBou
    nd="2"
54.         upperBound="-1" eType="#//TopologicalRelationship"/>
55.   </eClassifiers>
56.   <eClassifiers xsi:type="ecore:EEnum" name="Subordination">
57.     <eLiterals name="inner"/>
58.     <eLiterals name="external" value="1"/>
59.   </eClassifiers>
60.   <eClassifiers xsi:type="ecore:EEnum" name="LogicalOperation">
61.     <eLiterals name="AND"/>
62.     <eLiterals name="OR" value="1"/>
63.     <eLiterals name="XOR" value="2"/>
64.   </eClassifiers>
65. </ecore:EPackage>
```

IDM TOOLSET'S USE CASES EDITOR ARTIFACTS

Use Cases Editor Generation Model According to Eclipse EMF

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <genmodel:GenModel xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:ecore=
    "http://www.eclipse.org/emf/2002/Ecore"
3.      xmlns:genmodel="http://www.eclipse.org/emf/2002/GenModel" modelDirectory="/lv.rt
    u.ldk.idm.usecases/src" modelPluginID="lv.rtu.ldk.idm.usecases"
4.      modelName="UseCases" importerID="org.eclipse.emf.importer.ecore" complianceLevel
    ="6.0"
5.      copyrightFields="false">
6.    <foreignModel>../../Use%20Case%20Metamodel/UseCases.ecore</foreignModel>
7.    <genPackages prefix="Usecases" disposableProviderFactory="true" ecorePackage="UseC
    ases.ecore#/">
8.      <genEnums typeSafeEnumCompatible="false" ecoreEnum="UseCases.ecore#//Operation">

9.        <genEnumLiterals ecoreEnumLiteral="UseCases.ecore#//Operation/AND"/>
10.       <genEnumLiterals ecoreEnumLiteral="UseCases.ecore#//Operation/OR"/>
11.       <genEnumLiterals ecoreEnumLiteral="UseCases.ecore#//Operation/XOR"/>
12.     </genEnums>
13.     <genClasses ecoreClass="UseCases.ecore#//UseCases">
14.       <genFeatures property="None" children="true" createChild="true" ecoreFeature="
    ecore:EReference UseCases.ecore#//UseCases/actors"/>
15.       <genFeatures property="None" children="true" createChild="true" ecoreFeature="
    ecore:EReference UseCases.ecore#//UseCases/conditions"/>
16.       <genFeatures property="None" children="true" createChild="true" ecoreFeature="
    ecore:EReference UseCases.ecore#//UseCases/useCase"/>
17.       <genFeatures createChild="false" ecoreFeature="ecore:EAttribute UseCases.ecore
    #//UseCases/domain"/>
18.       <genFeatures createChild="false" ecoreFeature="ecore:EAttribute UseCases.ecore
    #//UseCases/scope"/>
19.       <genFeatures createChild="false" ecoreFeature="ecore:EAttribute UseCases.ecore
    #//UseCases/Ontology"/>
20.     </genClasses>
21.     <genClasses ecoreClass="UseCases.ecore#//UseCase">
22.       <genFeatures createChild="false" ecoreFeature="ecore:EAttribute UseCases.ecore
    #//UseCase/id"/>
23.       <genFeatures createChild="false" ecoreFeature="ecore:EAttribute UseCases.ecore
    #//UseCase/description"/>
24.       <genFeatures notify="false" createChild="false" propertySortChoices="true" eco
    reFeature="ecore:EReference UseCases.ecore#//UseCase/actors"/>
25.       <genFeatures property="None" children="true" createChild="true" ecoreFeature="
    ecore:EReference UseCases.ecore#//UseCase/mainScenario"/>
26.       <genFeatures property="None" children="true" createChild="true" ecoreFeature="
    ecore:EReference UseCases.ecore#//UseCase/extensions"/>
27.       <genFeatures property="None" children="true" createChild="true" ecoreFeature="
    ecore:EReference UseCases.ecore#//UseCase/subVariations"/>
28.       <genFeatures property="None" notify="false" createChild="false" ecoreFeature="
    ecore:EReference UseCases.ecore#//UseCase/useCases"/>
29.     </genClasses>
30.     <genClasses image="false" ecoreClass="UseCases.ecore#//SingleEvent">
31.       <genFeatures createChild="false" ecoreFeature="ecore:EAttribute UseCases.ecore
    #//SingleEvent/description"/>
32.       <genFeatures notify="false" createChild="false" propertySortChoices="true" eco
    reFeature="ecore:EReference UseCases.ecore#//SingleEvent/triggers"/>
```

```
      <genFeatures createChild="false" ecoreFeature="ecore:EAttribute UseCases.ecore
#//SingleEvent/reference"/>
    </genClasses>
    <genClasses ecoreClass="UseCases.ecore#//MainScenario">
      <genFeatures property="None" children="true" createChild="true" ecoreFeature="
ecore:EReference UseCases.ecore#//MainScenario/step"/>
      <genFeatures property="None" notify="false" createChild="false" ecoreFeature="
ecore:EReference UseCases.ecore#//MainScenario/mainScenarioUseCase"/>
    </genClasses>
    <genClasses ecoreClass="UseCases.ecore#//Extension">
      <genFeatures property="None" notify="false" createChild="false" ecoreFeature="
ecore:EReference UseCases.ecore#//Extension/extensionUseCase"/>
    </genClasses>
    <genClasses ecoreClass="UseCases.ecore#//SubVariation">
      <genFeatures property="None" notify="false" createChild="false" ecoreFeature="
ecore:EReference UseCases.ecore#//SubVariation/subVariationUseCase"/>
    </genClasses>
    <genClasses ecoreClass="UseCases.ecore#//DefaultEvent"/>
    <genClasses image="false" ecoreClass="UseCases.ecore#//AlternativeScenario">
      <genFeatures notify="false" createChild="false" propertySortChoices="true" eco
reFeature="ecore:EReference UseCases.ecore#//AlternativeScenario/reference"/>
      <genFeatures createChild="false" ecoreFeature="ecore:EAttribute UseCases.ecore
#//AlternativeScenario/id"/>
      <genFeatures property="None" children="true" createChild="true" ecoreFeature="
ecore:EReference UseCases.ecore#//AlternativeScenario/step"/>
    </genClasses>
    <genClasses ecoreClass="UseCases.ecore#//Actor">
      <genFeatures createChild="false" ecoreFeature="ecore:EAttribute UseCases.ecore
#//Actor/description"/>
    </genClasses>
    <genClasses image="false" ecoreClass="UseCases.ecore#//Scenario"/>
    <genClasses ecoreClass="UseCases.ecore#//AlternativeEvent"/>
    <genClasses ecoreClass="UseCases.ecore#//Actors">
      <genFeatures property="None" children="true" createChild="true" ecoreFeature="
ecore:EReference UseCases.ecore#//Actors/actor"/>
    </genClasses>
    <genClasses ecoreClass="UseCases.ecore#//Condition">
      <genFeatures createChild="false" ecoreFeature="ecore:EAttribute UseCases.ecore
#//Condition/id"/>
    </genClasses>
    <genClasses ecoreClass="UseCases.ecore#//Conditions">
      <genFeatures property="None" children="true" createChild="true" ecoreFeature="
ecore:EReference UseCases.ecore#//Conditions/condition"/>
      <genFeatures property="None" children="true" createChild="true" ecoreFeature="
ecore:EReference UseCases.ecore#//Conditions/compositeCondition"/>
    </genClasses>
    <genClasses ecoreClass="UseCases.ecore#//CompositeCondition">
      <genFeatures notify="false" createChild="false" propertySortChoices="true" eco
reFeature="ecore:EReference UseCases.ecore#//CompositeCondition/conditions"/>
      <genFeatures createChild="false" ecoreFeature="ecore:EAttribute UseCases.ecore
#//CompositeCondition/operation"/>
    </genClasses>
    <genClasses ecoreClass="UseCases.ecore#//Event">
      <genFeatures createChild="false" ecoreFeature="ecore:EAttribute UseCases.ecore
#//Event/id"/>
      <genFeatures notify="false" createChild="false" propertySortChoices="true" eco
reFeature="ecore:EReference UseCases.ecore#//Event/preconditions"/>
      <genFeatures notify="false" createChild="false" propertySortChoices="true" eco
reFeature="ecore:EReference UseCases.ecore#//Event/postconditions"/>
    </genClasses>
    <genClasses ecoreClass="UseCases.ecore#//SingleCondition">
      <genFeatures createChild="false" ecoreFeature="ecore:EAttribute UseCases.ecore
#//SingleCondition/description"/>
    </genClasses>
  </genPackages>
</genmodel:GenModel>
```

IDM TOOLSET'S TFM EDITOR ARTIFACTS

TFM Editor Generation Model According to Eclipse EMF

```
<?xml version="1.0" encoding="UTF-8"?>
<genmodel:GenModel xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore"
    xmlns:genmodel="http://www.eclipse.org/emf/2002/GenModel" modelDirectory="/lv.rtu.ldk.idm.tfm/src" modelPluginID="lv.rtu.ldk.idm.tfm"
    modelName="TFM" importerID="org.eclipse.emf.importer.ecore" complianceLevel="6.0"
    copyrightFields="false">
  <foreignModel>TFM.ecore</foreignModel>
  <genPackages prefix="Tfm" disposableProviderFactory="true" ecorePackage="TFM.ecore#/">
    <genEnums typeSafeEnumCompatible="false" ecoreEnum="TFM.ecore#//Subordination">
      <genEnumLiterals ecoreEnumLiteral="TFM.ecore#//Subordination/inner"/>
      <genEnumLiterals ecoreEnumLiteral="TFM.ecore#//Subordination/external"/>
    </genEnums>
    <genEnums typeSafeEnumCompatible="false" ecoreEnum="TFM.ecore#//LogicalOperation">
      <genEnumLiterals ecoreEnumLiteral="TFM.ecore#//LogicalOperation/AND"/>
      <genEnumLiterals ecoreEnumLiteral="TFM.ecore#//LogicalOperation/OR"/>
      <genEnumLiterals ecoreEnumLiteral="TFM.ecore#//LogicalOperation/XOR"/>
    </genEnums>
    <genClasses ecoreClass="TFM.ecore#//TFM">
      <genFeatures property="None" children="true" createChild="true" ecoreFeature="ecore:EReference TFM.ecore#//TFM/functionalFeatures"/>
      <genFeatures property="None" children="true" createChild="true" ecoreFeature="ecore:EReference TFM.ecore#//TFM/topologicalRelationships"/>
      <genFeatures property="None" children="true" createChild="true" ecoreFeature="ecore:EReference TFM.ecore#//TFM/logicalRelationships"/>
      <genFeatures property="None" children="true" createChild="true" ecoreFeature="ecore:EReference TFM.ecore#//TFM/actors"/>
      <genFeatures property="None" children="true" createChild="true" ecoreFeature="ecore:EReference TFM.ecore#//TFM/cycles"/>
    </genClasses>
    <genClasses ecoreClass="TFM.ecore#//FunctionalFeature">
      <genFeatures createChild="false" ecoreFeature="ecore:EAttribute TFM.ecore#//FunctionalFeature/id"/>
      <genFeatures createChild="false" ecoreFeature="ecore:EAttribute TFM.ecore#//FunctionalFeature/description"/>
      <genFeatures createChild="false" ecoreFeature="ecore:EAttribute TFM.ecore#//FunctionalFeature/action"/>
      <genFeatures createChild="false" ecoreFeature="ecore:EAttribute TFM.ecore#//FunctionalFeature/result"/>
      <genFeatures notify="false" createChild="false" propertySortChoices="true" ecoreFeature="ecore:EAttribute TFM.ecore#//FunctionalFeature/object"/>
      <genFeatures notify="false" createChild="false" propertySortChoices="true" ecoreFeature="ecore:EReference TFM.ecore#//FunctionalFeature/entity"/>
      <genFeatures createChild="false" ecoreFeature="ecore:EAttribute TFM.ecore#//FunctionalFeature/precond"/>
      <genFeatures createChild="false" ecoreFeature="ecore:EAttribute TFM.ecore#//FunctionalFeature/postcond"/>
      <genFeatures createChild="false" ecoreFeature="ecore:EAttribute TFM.ecore#//FunctionalFeature/subordination"/>
```

```
34.       <genFeatures createChild="false" ecoreFeature="ecore:EAttribute TFM.ecore#//Fu
    nctionalFeature/executorIsSystem"/>
35.     </genClasses>
36.     <genClasses ecoreClass="TFM.ecore#//Cycle">
37.       <genFeatures createChild="false" ecoreFeature="ecore:EAttribute TFM.ecore#//Cy
    cle/order"/>
38.       <genFeatures createChild="false" ecoreFeature="ecore:EAttribute TFM.ecore#//Cy
    cle/isMain"/>
39.       <genFeatures notify="false" createChild="false" propertySortChoices="true" eco
    reFeature="ecore:EReference TFM.ecore#//Cycle/functionalFeatures"/>
40.     </genClasses>
41.     <genClasses ecoreClass="TFM.ecore#//Actor">
42.       <genFeatures createChild="false" ecoreFeature="ecore:EAttribute TFM.ecore#//Ac
    tor/description"/>
43.     </genClasses>
44.     <genClasses ecoreClass="TFM.ecore#//TopologicalRelationship">
45.       <genFeatures createChild="false" ecoreFeature="ecore:EAttribute TFM.ecore#//To
    pologicalRelationship/id"/>
46.       <genFeatures notify="false" createChild="false" propertySortChoices="true" eco
    reFeature="ecore:EReference TFM.ecore#//TopologicalRelationship/source"/>
47.       <genFeatures notify="false" createChild="false" propertySortChoices="true" eco
    reFeature="ecore:EReference TFM.ecore#//TopologicalRelationship/target"/>
48.     </genClasses>
49.     <genClasses ecoreClass="TFM.ecore#//LogicalRelationship">
50.       <genFeatures createChild="false" ecoreFeature="ecore:EAttribute TFM.ecore#//Lo
    gicalRelationship/id"/>
51.       <genFeatures createChild="false" ecoreFeature="ecore:EAttribute TFM.ecore#//Lo
    gicalRelationship/operation"/>
52.       <genFeatures notify="false" createChild="false" propertySortChoices="true" eco
    reFeature="ecore:EReference TFM.ecore#//LogicalRelationship/relatedElements"/>
53.     </genClasses>
54.   </genPackages>
55. </genmodel:GenModel>
```

IDM TOOLSET'S TFM DIAGRAM TOOL ARTIFACTS

TFM Diagram Tool Graphical Definition Model According to Eclipse GMF

```
<?xml version="1.0" encoding="UTF-8"?>
<gmfgraph:Canvas
    xmi:version="2.0"
    xmlns:xmi="http://www.omg.org/XMI"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:gmfgraph="http://www.eclipse.org/gmf/2006/GraphicalDefinition"
    name="tfm">
  <figures
      name="Default">
    <figures
        xsi:type="gmfgraph:PolylineDecoration"
        name="CycleFunctionalFeaturesTargetDecoration"/>
    <figures
        xsi:type="gmfgraph:PolylineDecoration"
        name="LogicalRelationshipRelatedElementsTargetDecoration"/>
    <figures
        xsi:type="gmfgraph:PolylineDecoration"
        name="FunctionalFeatureEntityTargetDecoration"/>
    <descriptors
        name="CycleFigure">
      <actualFigure
          xsi:type="gmfgraph:Rectangle"
          name="CycleFigure">
        <layout
            xsi:type="gmfgraph:FlowLayout"/>
        <children
            xsi:type="gmfgraph:Label"
            name="CycleOrderFigure"
            text="<...>"/>
      </actualFigure>
      <accessors
          figure="//@figures.0/@descriptors.0/@actualFigure/@children.0"/>
    </descriptors>
    <descriptors
        name="CycleFunctionalFeaturesFigure">
      <actualFigure
          xsi:type="gmfgraph:PolylineConnection"
          name="CycleFunctionalFeaturesFigure"
          targetDecoration="//@figures.0/@figures.0"/>
    </descriptors>
    <descriptors
        name="LogicalRelationshipFigure">
      <actualFigure
          xsi:type="gmfgraph:Rectangle"
          name="LogicalRelationshipFigure">
        <layout
            xsi:type="gmfgraph:FlowLayout"/>
        <children
            xsi:type="gmfgraph:Label"
            name="LogicalRelationshipOperationFigure"
            text="<...>"/>
      </actualFigure>
      <accessors
```

```
            figure="//@figures.0/@descriptors.2/@actualFigure/@children.0"/>
      </descriptors>
      <descriptors
          name="LogicalRelationshipRelatedElementsFigure">
        <actualFigure
            xsi:type="gmfgraph:PolylineConnection"
            name="LogicalRelationshipRelatedElementsFigure"
            targetDecoration="//@figures.0/@figures.1"/>
      </descriptors>
      <descriptors
          name="FunctionalFeatureEntityFigure">
        <actualFigure
            xsi:type="gmfgraph:PolylineConnection"
            name="FunctionalFeatureEntityFigure"
            targetDecoration="//@figures.0/@figures.2"/>
      </descriptors>
      <descriptors
          name="ActorFigure">
        <actualFigure
            xsi:type="gmfgraph:Rectangle"
            name="ActorFigure"/>
      </descriptors>
    </figures>
    <nodes
        name="Cycle"
        figure="CycleFigure"/>
    <nodes
        name="LogicalRelationship"
        figure="LogicalRelationshipFigure"/>
    <nodes
        name="Actor"
        figure="ActorFigure"/>
    <connections
        name="CycleFunctionalFeatures"
        figure="CycleFunctionalFeaturesFigure"/>
    <connections
        name="LogicalRelationshipRelatedElements"
        figure="LogicalRelationshipRelatedElementsFigure"/>
    <connections
        name="FunctionalFeatureEntity"
        figure="FunctionalFeatureEntityFigure"/>
    <labels
        name="CycleOrder"
        figure="CycleFigure"
        accessor="//@figures.0/@descriptors.0/@accessors.0"/>
    <labels
              name="LogicalRelationshipOperation"
              figure="LogicalRelationshipFigure"
              accessor="//@figures.0/@descriptors.2/@accessors.0"/>
        </gmfgraph:Canvas>
```

TFM Diagram Tool Tooling Definition Model According to Eclipse GMF

```
<?xml version="1.0" encoding="UTF-8"?>
<gmftool:ToolRegistry
    xmi:version="2.0"
    xmlns:xmi="http://www.omg.org/XMI"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:gmftool="http://www.eclipse.org/gmf/2005/ToolDefinition">
  <palette
      title="tfmPalette">
    <tools
```

```
10.         xsi:type="gmftool:ToolGroup"
11.         title="tfm">
12.       <tools
13.           xsi:type="gmftool:CreationTool"
14.           title="Actor"
15.           description="Create new Actor">
16.         <smallIcon
17.             xsi:type="gmftool:DefaultImage"/>
18.         <largeIcon
19.             xsi:type="gmftool:DefaultImage"/>
20.       </tools>
21.       <tools
22.           xsi:type="gmftool:CreationTool"
23.           title="Functional Feature"
24.           description="Create new FunctionalFeature">
25.         <smallIcon
26.             xsi:type="gmftool:DefaultImage"/>
27.         <largeIcon
28.             xsi:type="gmftool:DefaultImage"/>
29.       </tools>
30.       <tools
31.           xsi:type="gmftool:CreationTool"
32.           title="Topological Relationship"
33.           description="Create new TopologicalRelationship">
34.         <smallIcon
35.             xsi:type="gmftool:BundleImage"
36.             path="icons/full/obj16/TopologicalRelationship.gif"
37.             bundle="lv.rtu.ldk.idm.tfm.edit"/>
38.         <largeIcon
39.             xsi:type="gmftool:DefaultImage"/>
40.       </tools>
41.       <tools
42.           xsi:type="gmftool:CreationTool"
43.           title="Cycle"
44.           description="Create new Cycle">
45.         <smallIcon
46.             xsi:type="gmftool:DefaultImage"/>
47.         <largeIcon
48.             xsi:type="gmftool:DefaultImage"/>
49.       </tools>
50.       <tools
51.           xsi:type="gmftool:CreationTool"
52.           title="Logical Relationship"
53.           description="Create new LogicalRelationship">
54.         <smallIcon
55.             xsi:type="gmftool:DefaultImage"/>
56.         <largeIcon
57.             xsi:type="gmftool:DefaultImage"/>
58.       </tools>
59.       <tools
60.           xsi:type="gmftool:CreationTool"
61.           title="Link Logical Relationship"
62.           description="Link Logical Relationship">
63.         <smallIcon
64.             xsi:type="gmftool:DefaultImage"/>
65.         <largeIcon
66.             xsi:type="gmftool:DefaultImage"/>
67.       </tools>
68.     </tools>
69.   </palette>
70. </gmftool:ToolRegistry>
```

TFM Diagram Tool Mapping Model According to Eclipse GMF

```
<?xml version="1.0" encoding="UTF-8"?>
<gmfmap:Mapping
    xmi:version="2.0"
    xmlns:xmi="http://www.omg.org/XMI"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore"
    xmlns:gmfmap="http://www.eclipse.org/gmf/2008/mappings"
    xmlns:gmftool="http://www.eclipse.org/gmf/2005/ToolDefinition">
  <nodes>
    <containmentFeature
        href="TFM.ecore#//TFM/topologicalRelationships"/>
    <ownedChild>
      <domainMetaElement
          href="TFM.ecore#//TopologicalRelationship"/>
      <labelMappings
          xsi:type="gmfmap:FeatureLabelMapping">
        <diagramLabel
            href="TFM1.gmfgraph#CycleOrder"/>
        <features
            href="TFM.ecore#//TopologicalRelationship/id"/>
      </labelMappings>
      <tool
          xsi:type="gmftool:CreationTool"
          href="TFM.gmftool#//@palette/@tools.0/@tools.0"/>
      <diagramNode
          href="TFM1.gmfgraph#Cycle"/>
    </ownedChild>
  </nodes>
  <nodes>
    <containmentFeature
        href="TFM.ecore#//TFM/logicalRelationships"/>
    <ownedChild>
      <domainMetaElement
          href="TFM.ecore#//LogicalRelationship"/>
      <labelMappings
          xsi:type="gmfmap:FeatureLabelMapping">
        <diagramLabel
            href="TFM1.gmfgraph#LogicalRelationshipOperation"/>
        <features
            href="TFM.ecore#//LogicalRelationship/id"/>
      </labelMappings>
      <tool
          xsi:type="gmftool:CreationTool"
          href="TFM.gmftool#//@palette/@tools.0/@tools.1"/>
      <diagramNode
          href="TFM1.gmfgraph#LogicalRelationship"/>
    </ownedChild>
  </nodes>
  <nodes>
    <containmentFeature
        href="TFM.ecore#//TFM/actors"/>
    <ownedChild>
      <domainMetaElement
          href="TFM.ecore#//Actor"/>
      <labelMappings
          xsi:type="gmfmap:FeatureLabelMapping">
        <features
            href="TFM.ecore#//Actor/description"/>
      </labelMappings>
      <tool
          xsi:type="gmftool:CreationTool"
          href="TFM.gmftool#//@palette/@tools.0/@tools.2"/>
      <diagramNode
          href="TFM1.gmfgraph#Actor"/>
```

```
    </ownedChild>
  </nodes>
  <nodes>
    <containmentFeature
        href="TFM.ecore#//TFM/cycles"/>
    <ownedChild>
      <domainMetaElement
          href="TFM.ecore#//Cycle"/>
      <tool
          xsi:type="gmftool:CreationTool"
          href="TFM.gmftool#//@palette/@tools.0/@tools.3"/>
      <diagramNode
          href="TFM1.gmfgraph#Cycle"/>
    </ownedChild>
  </nodes>
  <nodes>
    <containmentFeature
        href="TFM.ecore#//TFM/functionalFeatures"/>
    <ownedChild>
      <domainMetaElement
          href="TFM.ecore#//FunctionalFeature"/>
      <labelMappings
          xsi:type="gmfmap:FeatureLabelMapping">
        <diagramLabel
            href="TFM1.gmfgraph#CycleOrder"/>
        <features
            href="TFM.ecore#//FunctionalFeature/id"/>
      </labelMappings>
      <tool
          xsi:type="gmftool:CreationTool"
          href="TFM.gmftool#//@palette/@tools.0/@tools.4"/>
      <diagramNode
          href="TFM1.gmfgraph#Cycle"/>
    </ownedChild>
  </nodes>
        <links>
          <tool
              xsi:type="gmftool:CreationTool"
              href="TFM.gmftool#//@palette/@tools.0/@tools.0"/>
          <diagramLink
              href="TFM1.gmfgraph#LogicalRelationshipRelatedElements"/>
          <linkMetaFeature
              xsi:type="ecore:EReference"
              href="TFM.ecore#//LogicalRelationship/relatedElements"/>
        </links>
        <links>
          <tool
              xsi:type="gmftool:CreationTool"
              href="TFM.gmftool#//@palette/@tools.0/@tools.1"/>
          <diagramLink
              href="TFM1.gmfgraph#CycleFunctionalFeatures"/>
          <linkMetaFeature
              xsi:type="ecore:EReference"
              href="TFM.ecore#//Cycle/functionalFeatures"/>
        </links>
        <links>
          <tool
              xsi:type="gmftool:CreationTool"
              href="TFM.gmftool#//@palette/@tools.0/@tools.2"/>
          <diagramLink
              href="TFM1.gmfgraph#CycleFunctionalFeatures"/>
          <linkMetaFeature
              xsi:type="ecore:EReference"
              href="TFM.ecore#//TopologicalRelationship/source"/>
        </links>
        <links>
          <tool
              xsi:type="gmftool:CreationTool"
              href="TFM.gmftool#//@palette/@tools.0/@tools.3"/>
```

```
            <diagramLink
                href="TFM1.gmfgraph#CycleFunctionalFeatures"/>
            <linkMetaFeature
                xsi:type="ecore:EReference"
                href="TFM.ecore#//FunctionalFeature/entity"/>
          </links>
          <links>
            <tool
                xsi:type="gmftool:CreationTool"
                href="TFM.gmftool#//@palette/@tools.0/@tools.4"/>
            <diagramLink
                href="TFM1.gmfgraph#CycleFunctionalFeatures"/>
            <linkMetaFeature
                xsi:type="ecore:EReference"
                href="TFM.ecore#//TopologicalRelationship/target"/>
          </links>
          <diagram>
            <diagramCanvas
                href="TFM1.gmfgraph#tfm"/>
            <domainModel
                href="TFM.ecore#/"/>
            <domainMetaElement
                href="TFM.ecore#//TFM"/>
            <palette
                href="TFM.gmftool#//@palette"/>
          </diagram>
        </gmfmap:Mapping>
```

TFM Diagram Tool Diagram Editor Generation Model According to Eclipse GMF

```
<?xml version="1.0" encoding="UTF-8"?>
<gmfgen:GenEditorGenerator
    xmi:version="2.0"
    xmlns:xmi="http://www.omg.org/XMI"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:gmfgen="http://www.eclipse.org/gmf/2009/GenModel">
  <diagram
      visualID="1000"
      editPartClassName="TFMEditPart"
      itemSemanticEditPolicyClassName="TFMItemSemanticEditPolicy"
      canonicalEditPolicyClassName="TFMCanonicalEditPolicy"
      iconProviderPriority="Low"
      validationProviderPriority="Low">
    <diagramRunTimeClass
        href="../../../plugin/org.eclipse.gmf.runtime.notation/model/notation.genmodel#//notation/Diagram"/>
    <elementType
        xsi:type="gmfgen:MetamodelType"
        editHelperClassName="TFMEditHelper"/>
    <viewmap
        xsi:type="gmfgen:FigureViewmap"
        figureQualifiedClassName="org.eclipse.draw2d.FreeformLayer"/>
    <domainDiagramElement
        href="TFM.genmodel#//tfm/TFM"/>
    <topLevelNodes
        visualID="2001"
        editPartClassName="LogicalRelationshipEditPart"
        itemSemanticEditPolicyClassName="LogicalRelationshipItemSemanticEditPolicy"
        notationViewFactoryClassName="LogicalRelationshipViewFactory"
        canonicalEditPolicyClassName="LogicalRelationshipCanonicalEditPolicy"
        graphicalNodeEditPolicyClassName="LogicalRelationshipGraphicalNodeEditPolicy"
```

```
        createCommandClassName="LogicalRelationshipCreateCommand">
      <diagramRunTimeClass
          href="../../../plugin/org.eclipse.gmf.runtime.notation/model/notation.genmodel#//notation/Node"/>
      <elementType
          xsi:type="gmfgen:MetamodelType"
          editHelperClassName="LogicalRelationshipEditHelper"/>
      <viewmap
          xsi:type="gmfgen:InnerClassViewmap"
          className="LogicalRelationshipFigure"
          classBody="
/**
 * @generated
 */
public class LogicalRelationshipFigure extends org.eclipse.draw2d.Ellipse {


    /**
     * @generated
     */
    private org.eclipse.gmf.runtime.draw2d.ui.figures.WrappingLabel fFigureLogicalRelationshipOperation;


    /**
     * @generated
     */
    public LogicalRelationshipFigure() {

    org.eclipse.draw2d.GridLayout layoutThis = new org.eclipse.draw2d.GridLayout();

    layoutThis.numColumns = 1;
    layoutThis.makeColumnsEqualWidth = false;
    this.setLayoutManager(layoutThis);

        this.setPreferredSize(new org.eclipse.draw2d.geometry.Dimension(getMapMode().DPtoLP(30)
, getMapMode().DPtoLP(20)
));

this.setBorder(new org.eclipse.draw2d.MarginBorder(getMapMode().DPtoLP(0)
, getMapMode().DPtoLP(3)
, getMapMode().DPtoLP(0)
, getMapMode().DPtoLP(0)
));
        createContents();
    }
    /**
     * @generated
     */
    private void createContents(){


fFigureLogicalRelationshipOperation = new org.eclipse.gmf.runtime.draw2d.ui.figures.WrappingLabel();

fFigureLogicalRelationshipOperation.setText("<...>");

this.add(fFigureLogicalRelationshipOperation);


    }




    /**
     * @generated
```

```
95.      */
96.     public org.eclipse.gmf.runtime.draw2d.ui.figures.WrappingLabel getFigureLogicalR
    elationshipOperation() {
97.         return fFigureLogicalRelationshipOperation;
98.     }
99.
100.
101.       }
102.
103.       ">
104.                 <attributes
105.                     xsi:type="gmfgen:DefaultSizeAttributes"
106.                     width="30"
107.                     height="20"/>
108.               </viewmap>
109.               <modelFacet>
110.                 <metaClass
111.                     href="TFM.genmodel#//tfm/LogicalRelationship"/>
112.                 <containmentMetaFeature
113.                     href="TFM.genmodel#//tfm/TFM/logicalRelationships"/>
114.                 <childMetaFeature
115.                     href="TFM.genmodel#//tfm/TFM/logicalRelationships"/>
116.               </modelFacet>
117.               <labels
118.                   visualID="5001"
119.                   editPartClassName="LogicalRelationshipIdEditPart"
120.                   itemSemanticEditPolicyClassName="LogicalRelationshipIdItemSemanticE
    ditPolicy"
121.                   notationViewFactoryClassName="LogicalRelationshipIdViewFactory"
122.                   readOnly="true">
123.                 <diagramRunTimeClass
124.                     href="../../../plugin/org.eclipse.gmf.runtime.notation/model/nota
    tion.genmodel#//notation/Node"/>
125.                 <viewmap
126.                     xsi:type="gmfgen:ParentAssignedViewmap"
127.                     getterName="getFigureLogicalRelationshipOperation"
128.                     figureQualifiedClassName="org.eclipse.gmf.runtime.draw2d.ui.figur
    es.WrappingLabel"/>
129.                 <modelFacet
130.                     xsi:type="gmfgen:FeatureLabelModelFacet"
131.                     parser="//@labelParsers/@implementations.0"
132.                     viewPattern="">
133.                   <metaFeatures
134.                       href="TFM.genmodel#//tfm/LogicalRelationship/operation"/>
135.                 </modelFacet>
136.               </labels>
137.             </topLevelNodes>
138.             <topLevelNodes
139.                 visualID="2002"
140.                 editPartClassName="CycleEditPart"
141.                 itemSemanticEditPolicyClassName="CycleItemSemanticEditPolicy"
142.                 notationViewFactoryClassName="CycleViewFactory"
143.                 canonicalEditPolicyClassName="CycleCanonicalEditPolicy"
144.                 graphicalNodeEditPolicyClassName="CycleGraphicalNodeEditPolicy"
145.                 createCommandClassName="CycleCreateCommand">
146.               <diagramRunTimeClass
147.                   href="../../../plugin/org.eclipse.gmf.runtime.notation/model/notati
    on.genmodel#//notation/Node"/>
148.               <elementType
149.                   xsi:type="gmfgen:MetamodelType"
150.                   editHelperClassName="CycleEditHelper"/>
151.               <viewmap
152.                   xsi:type="gmfgen:InnerClassViewmap"
153.                   className="CycleFigure"
154.                   classBody="
155.       /**
156.        * @generated
157.        */
158.       public class CycleFigure extends org.eclipse.draw2d.Ellipse {
```

```
159.
160.
161.            /**
162.             * @generated
163.             */
164.            private org.eclipse.gmf.runtime.draw2d.ui.figures.WrappingLabel fFigureCy
    cleOrderFigure;
165.
166.
167.            /**
168.             * @generated
169.             */
170.            public CycleFigure() {
171.
172.            org.eclipse.draw2d.GridLayout layoutThis = new org.eclipse.draw2d.GridLay
    out();
173.            layoutThis.numColumns = 1;
174.            layoutThis.makeColumnsEqualWidth = false;
175.            this.setLayoutManager(layoutThis);
176.
177.                this.setPreferredSize(new org.eclipse.draw2d.geometry.Dimension(getMa
    pMode().DPtoLP(30)
178.        , getMapMode().DPtoLP(20)
179.        ));
180.                createContents();
181.            }
182.            /**
183.             * @generated
184.             */
185.            private void createContents(){
186.
187.
188.        fFigureCycleOrderFigure = new org.eclipse.gmf.runtime.draw2d.ui.figures.Wrapp
    ingLabel();
189.
190.        fFigureCycleOrderFigure.setText("<...>");
191.
192.        this.add(fFigureCycleOrderFigure);
193.
194.
195.            }
196.
197.
198.
199.
200.
201.            /**
202.             * @generated
203.             */
204.            public org.eclipse.gmf.runtime.draw2d.ui.figures.WrappingLabel getFigureC
    ycleOrderFigure() {
205.                return fFigureCycleOrderFigure;
206.            }
207.
208.
209.        }
210.
211.        ">
212.                <attributes
213.                    xsi:type="gmfgen:DefaultSizeAttributes"
214.                    width="30"
215.                    height="20"/>
216.              </viewmap>
217.              <modelFacet>
218.                <metaClass
219.                    href="TFM.genmodel#//tfm/Cycle"/>
220.                <containmentMetaFeature
221.                    href="TFM.genmodel#//tfm/TFM/cycles"/>
222.                <childMetaFeature
```

```
                    href="TFM.genmodel#//tfm/TFM/cycles"/>
              </modelFacet>
              <labels
                  visualID="5006"
                  editPartClassName="CycleOrderEditPart"
                  itemSemanticEditPolicyClassName="CycleOrderItemSemanticEditPolicy"
                  notationViewFactoryClassName="CycleOrderViewFactory"
                  elementIcon="true">
                <diagramRunTimeClass
                    href="../../../plugin/org.eclipse.gmf.runtime.notation/model/notation.genmodel#//notation/Node"/>
                <viewmap
                    xsi:type="gmfgen:ParentAssignedViewmap"
                    getterName="getFigureCycleOrderFigure"
                    figureQualifiedClassName="org.eclipse.gmf.runtime.draw2d.ui.figures.WrappingLabel"/>
                <modelFacet
                    xsi:type="gmfgen:FeatureLabelModelFacet"
                    parser="//@labelParsers/@implementations.1">
                  <metaFeatures
                      href="TFM.genmodel#//tfm/Cycle/order"/>
                </modelFacet>
              </labels>
            </topLevelNodes>
            <topLevelNodes
                visualID="2003"
                editPartClassName="ActorEditPart"
                itemSemanticEditPolicyClassName="ActorItemSemanticEditPolicy"
                notationViewFactoryClassName="ActorViewFactory"
                canonicalEditPolicyClassName="ActorCanonicalEditPolicy"
                graphicalNodeEditPolicyClassName="ActorGraphicalNodeEditPolicy"
                createCommandClassName="ActorCreateCommand">
              <diagramRunTimeClass
                  href="../../../plugin/org.eclipse.gmf.runtime.notation/model/notation.genmodel#//notation/Node"/>
              <elementType
                  xsi:type="gmfgen:MetamodelType"
                  editHelperClassName="ActorEditHelper"/>
              <viewmap
                  xsi:type="gmfgen:InnerClassViewmap"
                  className="ActorFigure"
                  classBody="
      /**
       * @generated
       */
      public class ActorFigure extends org.eclipse.draw2d.RectangleFigure {


          /**
           * @generated
           */
          private org.eclipse.gmf.runtime.draw2d.ui.figures.WrappingLabel fFigureActorDescriptionFigure;


          /**
           * @generated
           */
          public ActorFigure() {

          org.eclipse.draw2d.GridLayout layoutThis = new org.eclipse.draw2d.GridLayout();
          layoutThis.numColumns = 1;
          layoutThis.makeColumnsEqualWidth = false;
          this.setLayoutManager(layoutThis);

                  this.setBackgroundColor(THIS_BACK
      );
```

```
286.        this.setPreferredSize(new org.eclipse.draw2d.geometry.Dimension(getMapMode().
    DPtoLP(80)
287.        , getMapMode().DPtoLP(20)
288.        ));
289.                createContents();
290.            }
291.            /**
292.             * @generated
293.             */
294.            private void createContents(){
295.
296.
297.        fFigureActorDescriptionFigure = new org.eclipse.gmf.runtime.draw2d.ui.figures
    .WrappingLabel();
298.
299.        fFigureActorDescriptionFigure.setText("<...>");
300.
301.        this.add(fFigureActorDescriptionFigure);
302.
303.
304.            }
305.
306.
307.
308.
309.
310.            /**
311.             * @generated
312.             */
313.            public org.eclipse.gmf.runtime.draw2d.ui.figures.WrappingLabel getFigureA
    ctorDescriptionFigure() {
314.                return fFigureActorDescriptionFigure;
315.            }
316.
317.
318.        }
319.
320.        /**
321.         * @generated
322.         */
323.        static final org.eclipse.swt.graphics.Color THIS_BACK = new org.eclipse.swt.g
    raphics.Color(null, 213, 216, 247);
324.
325.        ">
326.                <attributes
327.                    xsi:type="gmfgen:StyleAttributes"
328.                    fixedBackground="true"/>
329.                <attributes
330.                    xsi:type="gmfgen:DefaultSizeAttributes"
331.                    width="80"
332.                    height="20"/>
333.              </viewmap>
334.              <modelFacet>
335.                <metaClass
336.                    href="TFM.genmodel#//tfm/Actor"/>
337.                <containmentMetaFeature
338.                    href="TFM.genmodel#//tfm/TFM/actors"/>
339.                <childMetaFeature
340.                    href="TFM.genmodel#//tfm/TFM/actors"/>
341.              </modelFacet>
342.              <labels
343.                  visualID="5002"
344.                  editPartClassName="ActorDescriptionEditPart"
345.                  itemSemanticEditPolicyClassName="ActorDescriptionItemSemanticEditPo
    licy"
346.                  notationViewFactoryClassName="ActorDescriptionViewFactory"
347.                  elementIcon="true">
348.                <diagramRunTimeClass
```

```
                    href="../../../plugin/org.eclipse.gmf.runtime.notation/model/notation.genmodel#//notation/Node"/>
                <viewmap
                    xsi:type="gmfgen:ParentAssignedViewmap"
                    getterName="getFigureActorDescriptionFigure"
                    figureQualifiedClassName="org.eclipse.gmf.runtime.draw2d.ui.figures.WrappingLabel"/>
                <modelFacet
                    xsi:type="gmfgen:FeatureLabelModelFacet"
                    parser="//@labelParsers/@implementations.1">
                  <metaFeatures
                     href="TFM.genmodel#//tfm/Actor/description"/>
                </modelFacet>
              </labels>
            </topLevelNodes>
            <topLevelNodes
                visualID="2005"
                editPartClassName="FunctionalFeatureEditPart"
                itemSemanticEditPolicyClassName="FunctionalFeatureItemSemanticEditPolicy"
                notationViewFactoryClassName="FunctionalFeatureViewFactory"
                canonicalEditPolicyClassName="FunctionalFeatureCanonicalEditPolicy"
                graphicalNodeEditPolicyClassName="FunctionalFeatureGraphicalNodeEditPolicy"
                createCommandClassName="FunctionalFeatureCreateCommand">
              <diagramRunTimeClass
                  href="../../../plugin/org.eclipse.gmf.runtime.notation/model/notation.genmodel#//notation/Node"/>
              <elementType
                  xsi:type="gmfgen:MetamodelType"
                  editHelperClassName="FunctionalFeatureEditHelper"/>
              <viewmap
                  xsi:type="gmfgen:InnerClassViewmap"
                  className="FunctionalFeatureFigure"
                  classBody="
        /**
         * @generated
         */
        public class FunctionalFeatureFigure extends org.eclipse.draw2d.RoundedRectangle {


            /**
             * @generated
             */
            private org.eclipse.gmf.runtime.draw2d.ui.figures.WrappingLabel fFigureFunctionalFeatureIdFigure;
            /**
             * @generated
             */
            private org.eclipse.gmf.runtime.draw2d.ui.figures.WrappingLabel fFigureFunctionalFeatureDescriptionFigure;


            /**
             * @generated
             */
            public FunctionalFeatureFigure() {

            org.eclipse.draw2d.GridLayout layoutThis = new org.eclipse.draw2d.GridLayout();
            layoutThis.numColumns = 2;
            layoutThis.makeColumnsEqualWidth = false;
            this.setLayoutManager(layoutThis);

                this.setCornerDimensions(new org.eclipse.draw2d.geometry.Dimension(getMapMode().DPtoLP(8)
        , getMapMode().DPtoLP(8)
        ));
```

```
            this.setBackgroundColor(THIS_BACK
        );
        this.setPreferredSize(new org.eclipse.draw2d.geometry.Dimension(getMapMode().
    DPtoLP(50)
        , getMapMode().DPtoLP(10)
        ));
                createContents();
            }
            /**
             * @generated
             */
            private void createContents(){


        fFigureFunctionalFeatureIdFigure = new org.eclipse.gmf.runtime.draw2d.ui.figu
    res.WrappingLabel();

        fFigureFunctionalFeatureIdFigure.setText("<...>");

        this.add(fFigureFunctionalFeatureIdFigure);



        fFigureFunctionalFeatureDescriptionFigure = new org.eclipse.gmf.runtime.draw2
    d.ui.figures.WrappingLabel();

        fFigureFunctionalFeatureDescriptionFigure.setText("<...>");

        this.add(fFigureFunctionalFeatureDescriptionFigure);


            }




            /**
             * @generated
             */
            public org.eclipse.gmf.runtime.draw2d.ui.figures.WrappingLabel getFigureF
    unctionalFeatureIdFigure() {
                return fFigureFunctionalFeatureIdFigure;
            }
            /**
             * @generated
             */
            public org.eclipse.gmf.runtime.draw2d.ui.figures.WrappingLabel getFigureF
    unctionalFeatureDescriptionFigure() {
                return fFigureFunctionalFeatureDescriptionFigure;
            }


        }

        /**
         * @generated
         */
        static final org.eclipse.swt.graphics.Color THIS_BACK = new org.eclipse.swt.g
    raphics.Color(null, 220, 247, 221);

        ">
                <attributes
                    xsi:type="gmfgen:StyleAttributes"
                    fixedBackground="true"/>
                <attributes
                    xsi:type="gmfgen:DefaultSizeAttributes"
                    width="50"
                    height="10"/>
```

```
471.             </viewmap>
472.             <modelFacet>
473.               <metaClass
474.                   href="TFM.genmodel#//tfm/FunctionalFeature"/>
475.               <containmentMetaFeature
476.                   href="TFM.genmodel#//tfm/TFM/functionalFeatures"/>
477.               <childMetaFeature
478.                   href="TFM.genmodel#//tfm/TFM/functionalFeatures"/>
479.             </modelFacet>
480.             <labels
481.                 visualID="5004"
482.                 editPartClassName="FunctionalFeatureIdEditPart"
483.                 itemSemanticEditPolicyClassName="FunctionalFeatureIdItemSemanticEdi
    tPolicy"
484.                 notationViewFactoryClassName="FunctionalFeatureIdViewFactory"
485.                 elementIcon="true">
486.               <diagramRunTimeClass
487.                   href="../../../plugin/org.eclipse.gmf.runtime.notation/model/nota
    tion.genmodel#//notation/Node"/>
488.               <viewmap
489.                   xsi:type="gmfgen:ParentAssignedViewmap"
490.                   getterName="getFigureFunctionalFeatureIdFigure"
491.                   figureQualifiedClassName="org.eclipse.gmf.runtime.draw2d.ui.figur
    es.WrappingLabel"/>
492.               <modelFacet
493.                   xsi:type="gmfgen:FeatureLabelModelFacet"
494.                   parser="//@labelParsers/@implementations.1">
495.                 <metaFeatures
496.                     href="TFM.genmodel#//tfm/FunctionalFeature/id"/>
497.               </modelFacet>
498.             </labels>
499.             <labels
500.                 visualID="5005"
501.                 editPartClassName="FunctionalFeatureDescriptionEditPart"
502.                 itemSemanticEditPolicyClassName="FunctionalFeatureDescriptionItemSe
    manticEditPolicy"
503.                 notationViewFactoryClassName="FunctionalFeatureDescriptionViewFacto
    ry">
504.               <diagramRunTimeClass
505.                   href="../../../plugin/org.eclipse.gmf.runtime.notation/model/nota
    tion.genmodel#//notation/Node"/>
506.               <viewmap
507.                   xsi:type="gmfgen:ParentAssignedViewmap"
508.                   getterName="getFigureFunctionalFeatureDescriptionFigure"
509.                   figureQualifiedClassName="org.eclipse.gmf.runtime.draw2d.ui.figur
    es.WrappingLabel"/>
510.               <modelFacet
511.                   xsi:type="gmfgen:FeatureLabelModelFacet"
512.                   parser="//@labelParsers/@implementations.1">
513.                 <metaFeatures
514.                     href="TFM.genmodel#//tfm/FunctionalFeature/description"/>
515.               </modelFacet>
516.             </labels>
517.           </topLevelNodes>
518.           <links
519.               visualID="4009"
520.               editPartClassName="LogicalRelationshipRelatedElementsEditPart"
521.               itemSemanticEditPolicyClassName="LogicalRelationshipRelatedElementsIt
    emSemanticEditPolicy"
522.               notationViewFactoryClassName="LogicalRelationshipRelatedElementsViewF
    actory"
523.               createCommandClassName="LogicalRelationshipRelatedElementsCreateComma
    nd"
524.               reorientCommandClassName="LogicalRelationshipRelatedElementsReorientC
    ommand">
525.             <diagramRunTimeClass
526.                 href="../../../plugin/org.eclipse.gmf.runtime.notation/model/notati
    on.genmodel#//notation/Edge"/>
527.             <elementType
```

```
528.                 xsi:type="gmfgen:SpecializationType"/>
529.             <viewmap
530.                 xsi:type="gmfgen:FigureViewmap"
531.                 figureQualifiedClassName="org.eclipse.gmf.runtime.draw2d.ui.figures
    .PolylineConnectionEx"/>
532.             <modelFacet
533.                 xsi:type="gmfgen:FeatureLinkModelFacet">
534.               <metaFeature
535.                   href="TFM.genmodel#//tfm/LogicalRelationship/relatedElements"/>
536.             </modelFacet>
537.           </links>
538.           <links
539.               visualID="4004"
540.               editPartClassName="FunctionalFeatureEntityEditPart"
541.               itemSemanticEditPolicyClassName="FunctionalFeatureEntityItemSemanticE
    ditPolicy"
542.               notationViewFactoryClassName="FunctionalFeatureEntityViewFactory"
543.               createCommandClassName="FunctionalFeatureEntityCreateCommand"
544.               reorientCommandClassName="FunctionalFeatureEntityReorientCommand">
545.             <diagramRunTimeClass
546.                 href="../../../plugin/org.eclipse.gmf.runtime.notation/model/notati
    on.genmodel#//notation/Edge"/>
547.             <elementType
548.                 xsi:type="gmfgen:SpecializationType"/>
549.             <viewmap
550.                 xsi:type="gmfgen:InnerClassViewmap"
551.                 className="FunctionalFeatureEntityFigure"
552.                 classBody="
553.       /**
554.        * @generated
555.        */
556.       public class FunctionalFeatureEntityFigure extends org.eclipse.gmf.runtime.dr
    aw2d.ui.figures.PolylineConnectionEx {
557.
558.
559.
560.
561.           /**
562.            * @generated
563.            */
564.           public FunctionalFeatureEntityFigure() {
565.               this.setLineStyle(org.eclipse.draw2d.Graphics.LINE_DASH);
566.
567.               setTargetDecoration(createTargetDecoration());
568.           }
569.
570.           /**
571.            * @generated
572.            */
573.           private org.eclipse.draw2d.RotatableDecoration createTargetDecoration() {

574.               org.eclipse.draw2d.PolylineDecoration df = new org.eclipse.draw2d.Pol
    ylineDecoration();
575.               return df;
576.           }
577.
578.
579.
580.
581.       }
582.
583.       "/>
584.             <modelFacet
585.                 xsi:type="gmfgen:FeatureLinkModelFacet">
586.               <metaFeature
587.                   href="TFM.genmodel#//tfm/FunctionalFeature/entity"/>
588.             </modelFacet>
589.           </links>
590.           <links
```

```
591.                visualID="4005"
592.                editPartClassName="CycleFunctionalFeaturesEditPart"
593.                itemSemanticEditPolicyClassName="CycleFunctionalFeaturesItemSemanticE
    ditPolicy"
594.                notationViewFactoryClassName="CycleFunctionalFeaturesViewFactory"
595.                createCommandClassName="CycleFunctionalFeaturesCreateCommand"
596.                reorientCommandClassName="CycleFunctionalFeaturesReorientCommand">
597.              <diagramRunTimeClass
598.                  href="../../../plugin/org.eclipse.gmf.runtime.notation/model/notati
    on.genmodel#//notation/Edge"/>
599.              <elementType
600.                  xsi:type="gmfgen:SpecializationType"/>
601.              <viewmap
602.                  xsi:type="gmfgen:InnerClassViewmap"
603.                  className="CycleFunctionalFeaturesFigure"
604.                  classBody="
605.       /**
606.        * @generated
607.        */
608.       public class CycleFunctionalFeaturesFigure extends org.eclipse.gmf.runtime.dr
    aw2d.ui.figures.PolylineConnectionEx {
609.
610.
611.
612.
613.           /**
614.            * @generated
615.            */
616.           public CycleFunctionalFeaturesFigure() {
617.               this.setLineStyle(org.eclipse.draw2d.Graphics.LINE_DASH);
618.
619.               setTargetDecoration(createTargetDecoration());
620.           }
621.
622.           /**
623.            * @generated
624.            */
625.           private org.eclipse.draw2d.RotatableDecoration createTargetDecoration() {

626.               org.eclipse.draw2d.PolylineDecoration df = new org.eclipse.draw2d.Pol
    ylineDecoration();
627.               return df;
628.           }
629.
630.
631.
632.
633.       }
634.
635.       "/>
636.             <modelFacet
637.                 xsi:type="gmfgen:FeatureLinkModelFacet">
638.               <metaFeature
639.                   href="TFM.genmodel#//tfm/Cycle/functionalFeatures"/>
640.             </modelFacet>
641.           </links>
642.           <links
643.               visualID="4007"
644.               editPartClassName="TopologicalRelationshipEditPart"
645.               itemSemanticEditPolicyClassName="TopologicalRelationshipItemSemanticE
    ditPolicy"
646.               notationViewFactoryClassName="TopologicalRelationshipViewFactory"
647.               createCommandClassName="TopologicalRelationshipCreateCommand"
648.               reorientCommandClassName="TopologicalRelationshipReorientCommand">
649.             <diagramRunTimeClass
650.                 href="../../../plugin/org.eclipse.gmf.runtime.notation/model/notati
    on.genmodel#//notation/Edge"/>
651.             <elementType
652.                 xsi:type="gmfgen:MetamodelType"
```

```
                editHelperClassName="TopologicalRelationshipEditHelper"/>
            <viewmap
                xsi:type="gmfgen:InnerClassViewmap"
                className="TopologicalRelationshipFigure"
                classBody="
       /**
        * @generated
        */
       public class TopologicalRelationshipFigure extends org.eclipse.gmf.runtime.dr
   aw2d.ui.figures.PolylineConnectionEx {


           /**
            * @generated
            */
           public TopologicalRelationshipFigure() {
               this.setLineWidth(2);

               setTargetDecoration(createTargetDecoration());
           }

           /**
            * @generated
            */
           private org.eclipse.draw2d.RotatableDecoration createTargetDecoration() {

               org.eclipse.draw2d.PolylineDecoration df = new org.eclipse.draw2d.Pol
   ylineDecoration();
       df.setLineWidth(2);
               return df;
           }


       }

       ">
               <attributes
                   xsi:type="gmfgen:StyleAttributes"
                   fixedForeground="true"/>
             </viewmap>
             <modelFacet
                 xsi:type="gmfgen:TypeLinkModelFacet">
               <metaClass
                   href="TFM.genmodel#//tfm/TopologicalRelationship"/>
               <containmentMetaFeature
                   href="TFM.genmodel#//tfm/TFM/topologicalRelationships"/>
               <childMetaFeature
                   href="TFM.genmodel#//tfm/TFM/topologicalRelationships"/>
               <sourceMetaFeature
                   href="TFM.genmodel#//tfm/TopologicalRelationship/source"/>
               <targetMetaFeature
                   href="TFM.genmodel#//tfm/TopologicalRelationship/target"/>
             </modelFacet>
           </links>
           <palette>
             <groups
                 title="tfm">
               <entries
                   xsi:type="gmfgen:ToolEntry"
                   title="Actor"
                   description="Create new Actor"
                   genNodes="//@diagram/@topLevelNodes.2"/>
               <entries
                   xsi:type="gmfgen:ToolEntry"
                   title="Functional Feature"
```

```
719.                    description="Create new FunctionalFeature"
720.                    genNodes="//@diagram/@topLevelNodes.3"/>
721.                <entries
722.                    xsi:type="gmfgen:ToolEntry"
723.                    title="Topological Relationship"
724.                    description="Create new TopologicalRelationship"
725.                    smallIconPath="/lv.rtu.ldk.idm.tfm.edit/icons/full/obj16/Topologi
    calRelationship.gif"
726.                    genLinks="//@diagram/@links.3"/>
727.                <entries
728.                    xsi:type="gmfgen:ToolEntry"
729.                    title="Cycle"
730.                    description="Create new Cycle"
731.                    genNodes="//@diagram/@topLevelNodes.1"/>
732.                <entries
733.                    xsi:type="gmfgen:ToolEntry"
734.                    title="Logical Relationship"
735.                    description="Create new LogicalRelationship"
736.                    genNodes="//@diagram/@topLevelNodes.0"/>
737.                <entries
738.                    xsi:type="gmfgen:ToolEntry"
739.                    title="Link Logical Relationship"
740.                    description="Link Logical Relationship"
741.                    genLinks="//@diagram/@links.0"/>
742.              </groups>
743.            </palette>
744.            <preferencePages
745.                xsi:type="gmfgen:GenStandardPreferencePage"
746.                iD="lv.rtu.ldk.idm.tfm.diagram.general"
747.                name="TFM Diagram">
748.              <children
749.                  xsi:type="gmfgen:GenStandardPreferencePage"
750.                  iD="lv.rtu.ldk.idm.tfm.diagram.appearance"
751.                  name="Appearance"
752.                  kind="Appearance"/>
753.              <children
754.                  xsi:type="gmfgen:GenStandardPreferencePage"
755.                  iD="lv.rtu.ldk.idm.tfm.diagram.connections"
756.                  name="Connections"
757.                  kind="Connections"/>
758.              <children
759.                  xsi:type="gmfgen:GenStandardPreferencePage"
760.                  iD="lv.rtu.ldk.idm.tfm.diagram.printing"
761.                  name="Printing"
762.                  kind="Printing"/>
763.              <children
764.                  xsi:type="gmfgen:GenStandardPreferencePage"
765.                  iD="lv.rtu.ldk.idm.tfm.diagram.rulersAndGrid"
766.                  name="Rulers And Grid"
767.                  kind="RulersAndGrid"/>
768.            </preferencePages>
769.          </diagram>
770.          <plugin>
771.            <requiredPlugins>org.eclipse.gmf.tooling.runtime</requiredPlugins>
772.            <requiredPlugins>org.eclipse.draw2d</requiredPlugins>
773.            <requiredPlugins>org.eclipse.gmf.runtime.draw2d.ui</requiredPlugins>
774.          </plugin>
775.          <editor/>
776.          <navigator>
777.            <childReferences
778.                child="//@diagram"/>
779.            <childReferences
780.                parent="//@diagram"
781.                child="//@diagram/@topLevelNodes.0"/>
782.            <childReferences
783.                parent="//@diagram"
784.                child="//@diagram/@topLevelNodes.1"/>
785.            <childReferences
786.                parent="//@diagram"
```

```
787.                child="//@diagram/@topLevelNodes.2"/>
788.            <childReferences
789.                parent="//@diagram"
790.                child="//@diagram/@topLevelNodes.3"/>
791.            <childReferences
792.                parent="//@diagram"
793.                child="//@diagram/@links.0"
794.                groupName="links"
795.                groupIcon="icons/linksNavigatorGroup.gif"/>
796.            <childReferences
797.                parent="//@diagram/@links.0"
798.                child="//@diagram/@topLevelNodes.0"
799.                referenceType="in_source"
800.                groupName="source"
801.                groupIcon="icons/linkSourceNavigatorGroup.gif"/>
802.            <childReferences
803.                parent="//@diagram/@topLevelNodes.0"
804.                child="//@diagram/@links.0"
805.                referenceType="out_target"
806.                groupName="outgoing links"
807.                groupIcon="icons/outgoingLinksNavigatorGroup.gif"/>
808.            <childReferences
809.                parent="//@diagram"
810.                child="//@diagram/@links.1"
811.                groupName="links"
812.                groupIcon="icons/linksNavigatorGroup.gif"/>
813.            <childReferences
814.                parent="//@diagram/@links.1"
815.                child="//@diagram/@topLevelNodes.2"
816.                referenceType="out_target"
817.                groupName="target"
818.                groupIcon="icons/linkTargetNavigatorGroup.gif"/>
819.            <childReferences
820.                parent="//@diagram/@topLevelNodes.2"
821.                child="//@diagram/@links.1"
822.                referenceType="in_source"
823.                groupName="incoming links"
824.                groupIcon="icons/incomingLinksNavigatorGroup.gif"/>
825.            <childReferences
826.                parent="//@diagram/@links.1"
827.                child="//@diagram/@topLevelNodes.3"
828.                referenceType="in_source"
829.                groupName="source"
830.                groupIcon="icons/linkSourceNavigatorGroup.gif"/>
831.            <childReferences
832.                parent="//@diagram/@topLevelNodes.3"
833.                child="//@diagram/@links.1"
834.                referenceType="out_target"
835.                groupName="outgoing links"
836.                groupIcon="icons/outgoingLinksNavigatorGroup.gif"/>
837.            <childReferences
838.                parent="//@diagram"
839.                child="//@diagram/@links.2"
840.                groupName="links"
841.                groupIcon="icons/linksNavigatorGroup.gif"/>
842.            <childReferences
843.                parent="//@diagram/@links.2"
844.                child="//@diagram/@topLevelNodes.3"
845.                referenceType="out_target"
846.                groupName="target"
847.                groupIcon="icons/linkTargetNavigatorGroup.gif"/>
848.            <childReferences
849.                parent="//@diagram/@topLevelNodes.3"
850.                child="//@diagram/@links.2"
851.                referenceType="in_source"
852.                groupName="incoming links"
853.                groupIcon="icons/incomingLinksNavigatorGroup.gif"/>
854.            <childReferences
855.                parent="//@diagram/@links.2"
```

```
            child="//@diagram/@topLevelNodes.1"
            referenceType="in_source"
            groupName="source"
            groupIcon="icons/linkSourceNavigatorGroup.gif"/>
        <childReferences
            parent="//@diagram/@topLevelNodes.1"
            child="//@diagram/@links.2"
            referenceType="out_target"
            groupName="outgoing links"
            groupIcon="icons/outgoingLinksNavigatorGroup.gif"/>
        <childReferences
            parent="//@diagram"
            child="//@diagram/@links.3"
            groupName="links"
            groupIcon="icons/linksNavigatorGroup.gif"/>
        <childReferences
            parent="//@diagram/@links.3"
            child="//@diagram/@topLevelNodes.3"
            referenceType="out_target"
            groupName="target"
            groupIcon="icons/linkTargetNavigatorGroup.gif"/>
        <childReferences
            parent="//@diagram/@topLevelNodes.3"
            child="//@diagram/@links.3"
            referenceType="in_source"
            groupName="incoming links"
            groupIcon="icons/incomingLinksNavigatorGroup.gif"/>
        <childReferences
            parent="//@diagram/@links.3"
            child="//@diagram/@topLevelNodes.3"
            referenceType="in_source"
            groupName="source"
            groupIcon="icons/linkSourceNavigatorGroup.gif"/>
        <childReferences
            parent="//@diagram/@topLevelNodes.3"
            child="//@diagram/@links.3"
            referenceType="out_target"
            groupName="outgoing links"
            groupIcon="icons/outgoingLinksNavigatorGroup.gif"/>
    </navigator>
    <diagramUpdater/>
    <propertySheet>
      <tabs
          xsi:type="gmfgen:GenStandardPropertyTab"
          iD="appearance"/>
      <tabs
          xsi:type="gmfgen:GenStandardPropertyTab"
          iD="diagram"/>
      <tabs
          xsi:type="gmfgen:GenCustomPropertyTab"
          iD="domain"
          label="Core">
        <filter
            xsi:type="gmfgen:TypeTabFilter">
          <types>org.eclipse.gmf.runtime.notation.View</types>
          <types>org.eclipse.gef.EditPart</types>
          <generatedTypes>abstractNavigatorItem</generatedTypes>
        </filter>
      </tabs>
    </propertySheet>
    <domainGenModel
        href="TFM.genmodel#/"/>
    <labelParsers
        extensibleViaService="true">
      <implementations
          xsi:type="gmfgen:PredefinedEnumParser"
          uses="//@diagram/@topLevelNodes.0/@labels.0/@modelFacet"/>
      <implementations
          xsi:type="gmfgen:PredefinedParser"
```

```
925.              uses="//@diagram/@topLevelNodes.1/@labels.0/@modelFacet //@diagram/@t
    opLevelNodes.2/@labels.0/@modelFacet //@diagram/@topLevelNodes.3/@labels.0/@modelFac
    et //@diagram/@topLevelNodes.3/@labels.1/@modelFacet"/>
926.          </labelParsers>
927.          <contextMenus
928.              context="//@diagram">
929.            <items
930.                xsi:type="gmfgen:LoadResourceAction"/>
931.          </contextMenus>
932.        </gmfgen:GenEditorGenerator>
```

IDM TOOLSET'S USE CASES TO TFM TRANSFORMATION TOOL ARTIFACTS

Use Cases to TFM Model Transformation Defined with QVTo Language

```
1.  import lv.rtu.ldk.idm.usecases2tfm.BlackBoxLibrary;
2.
3.  modeltype UseCases "strict" uses 'http://lv/rtu/ldk/idm/usecases/1.0';
4.  modeltype TFM "strict" uses 'http://ldi.rtu.lv/tfm/1.0';
5.
6.  transformation UseCases2TFM(in useCases : UseCases, out TFM);
7.
8.  main() {
9.      useCases.rootObjects()[UseCases::UseCases]-> map useCases2TFM();
10. }
11.
12. //Main mapping UseCases to TFM
13. mapping UseCases::UseCases::useCases2TFM() : TFM::TFM {
14.     var resolvedDescription : String;
15.     var resolvedEnity : String;
16.     var referencedSteps : Set(UseCases::SingleEvent);
17.
18.     var tempStep : UseCases::SingleEvent;
19.     var stepBuffer : OrderedSet(UseCases::SingleEvent);
20.     var sameSteps : OrderedSet(UseCases::SingleEvent);
21.
22.     //Preprocessing the Use Cases model
23.
24.     //Build step buffer with resolved references
25.     //sorting here is done for consistency reasons, so that references don't change

26.     self.allSubobjectsOfKind(SingleEvent)[UseCases::SingleEvent]->sortedBy(id)-
    >forEach(step) {
27.
28.         tempStep := step;
29.
30.         //Look for reference
31.         sameSteps := stepBuffer[UseCases::SingleEvent]->
32.             select(buffer | buffer.description = step.description and (buffer.refere
    nce.oclIsUndefined() or buffer.reference = ""));
33.
34.         //If reference found add to the buffer with a reference
35.         if (sameSteps->size() = 1) then {
36.             tempStep.reference := sameSteps->at(1).id;
37.
38.             log("Reference identified: "+tempStep.reference+" = "+tempStep.id);
39.         } endif;
40.
41.         stepBuffer += tempStep;
42.     };
43.
44.     //Write references back to the source model
45.     self.allSubobjectsOfKind(SingleEvent)[UseCases::SingleEvent]->forEach(step) {
46.         step.reference := stepBuffer-
    >selectOne(buffer | buffer.id = step.id).reference;
47.     };
48.
```

```
49.     //Create Actors
50.     self.actors.actor->forEach(a){
51.         actors += a.map actor2actor();
52.     };
53.
54.     //Sort for convenience
55.     actors := actors->sortedBy(description);
56.
57.     //Create Functional Features
58.     self.allSubobjectsOfKind(SingleEvent)[UseCases::SingleEvent]->forEach(step) {
59.         resolvedDescription := parseStepForDescription(step.description);
60.         resolvedEnity := parseStepForEntity(step.description);
61.
62.         //If step is referenced no need for a seperate Functional Feature (check als
    o if it is empty)
63.         if (step.reference.oclIsUndefined() or step.reference = "") then {
64.
65.             //Get referenced steps for conditions
66.             referencedSteps := step.getReferencedSteps(self);
67.
68.             //Create functional features
69.             functionalFeatures += step.map step2functionalFeature(actors, resolvedEn
    ity, resolvedDescription, referencedSteps);
70.
71.         } endif;
72.     };
73.
74.     //Sort for convenience
75.     functionalFeatures := functionalFeatures->sortedBy(id);
76.
77.     //Functional Relationships
78.     //According to scenarios
79.     self.useCase->forEach(useCase) {
80.         useCase.allSubobjectsOfKind(Scenario)[UseCases::Scenario]-
    >forEach(scenario) {
81.             topologicalRelationships +=
82.                 scenario.getTopologicalRelationships(functionalFeatures, useCase);
83.         };
84.     };
85.
86.     //Resolve triggers
87.     self.allSubobjectsOfKind(SingleEvent)[UseCases::SingleEvent]->forEach(step) {
88.
89.         step.triggers->forEach(trigger) {
90.             topologicalRelationships +=
91.                 map mapTopologicalRelationship(functionalFeatures, trigger.oclAsType
    (SingleEvent), step)
92.         };
93.
94.     };
95.
96.     //Sort for convenience
97.     topologicalRelationships := topologicalRelationships->sortedBy(id);
98.
99.     //Remove duplicate topological relationships
100.          var uniqueTopologicalRelationships : Set(TFM::TopologicalRelationship);
101.
102.          topologicalRelationships->forEach(uniqueTopologicalRelationship) {
103.              var foundTR : TFM::TopologicalRelationship;
104.              foundTR := uniqueTopologicalRelationships-
    >selectOne(tr | tr.id = uniqueTopologicalRelationship.id);
105.
106.              if (foundTR->isEmpty()) then{
107.                  uniqueTopologicalRelationships += uniqueTopologicalRelationship;

108.              } endif;
109.
110.          };
111.
```

```
            topologicalRelationships := uniqueTopologicalRelationships;

        }

        //Actor to Actor
        mapping UseCases::Actor::actor2actor() : TFM::Actor {
            description := self.description;
            log("Creating Actor: " + self.description);
        }

        //Resolve reference id
        helper resolveStepReference(ffs : OrderedSet(TFM::FunctionalFeature),
                    resolvedDescription : String) : String {

            var foundId : String := "";
            var foundFFs : OrderedSet(TFM::FunctionalFeature);

            foundFFs := ffs->select(ff | ff.description = resolvedDescription);

            if (foundFFs->size() > 0) then {
                foundId := foundFFs->at(1).id;
            } endif;

            return foundId;
        }

        //Get referenced steps
        helper UseCases::SingleEvent::getReferencedSteps(ucs : UseCases::UseCases) :
    Set(UseCases::SingleEvent) {
            var steps : Set(UseCases::SingleEvent);

            steps := ucs.allSubobjectsOfKind(SingleEvent)[UseCases::SingleEvent]-
    >asSet();
            return steps-
    >select(step | step.description = self.description and step.id != self.id);
        }

        //Single Event to Functional Feature
        mapping UseCases::SingleEvent::step2functionalFeature(actors : Set(TFM::Actor
    ),
                    resolvedEnity : String, resolvedDescription : String,
                    referencedSteps : Set(UseCases::SingleEvent)) : TFM::FunctionalFe
    ature {

            id := self.id;
            description := resolvedDescription;
            precond := self.getMergedPreConditions(referencedSteps);
            postcond := self.getMergedPostConditions(referencedSteps);

            entity := actors->selectOne(actor | actor.description = resolvedEnity);

            log("Creating Functional Feature: " + self.id.toString());
        }

        //Scenarios to Topological Relationships
        helper UseCases::Scenario::getTopologicalRelationships(ffs : Set(TFM::Functio
    nalFeature),
                useCase : UseCases::UseCase) : Set(TFM::TopologicalRelationship) {

            var topRels : Set(TFM::TopologicalRelationship);

            switch {
                case(self.oclIsTypeOf(MainScenario)) {
                    topRels := self.oclAsType(MainScenario).mainScenario2topRels(ffs)
    ;
                }
                case(self.oclIsTypeOf(Extension) or self.oclIsTypeOf(SubVariation)) {
```

```
                    topRels := self.oclAsType(AlternativeScenario).altScenario2topRels(ffs, useCase);
                }
            };

            return topRels;
        }

        //Main Scenario to Topological Relationships
        helper UseCases::MainScenario::mainScenario2topRels(ffs : Set(TFM::FunctionalFeature)) : Set(TFM::TopologicalRelationship) {
            var stepNumber : Integer = 0;
            var nextStep : UseCases::DefaultEvent;
            var topRels : Set(TFM::TopologicalRelationship);

            self.step->forEach (currentStep) {
                stepNumber := stepNumber + 1;
                nextStep := self.step->at(stepNumber+1);

                if (self.step->size() != stepNumber) then {
                    topRels += map mapTopologicalRelationship(ffs, currentStep, nextStep);
                } endif;
            };

            return topRels;
        }

        //Extension to Topological Relationships
        helper UseCases::AlternativeScenario::altScenario2topRels(ffs : Set(TFM::FunctionalFeature),
                    useCase : UseCases::UseCase) : Set(TFM::TopologicalRelationship) {

            var stepNumber : Integer = 0;
            var nextStep : UseCases::AlternativeEvent;
            var previousDefaultStep : UseCases::DefaultEvent;
            var topRels : Set(TFM::TopologicalRelationship);

            self.step->forEach (currentStep) {
                stepNumber := stepNumber + 1;
                nextStep := self.step->at(stepNumber+1);

                //Get source topological relationship according to alternative scenario
                if (stepNumber = 1) then {
                    if (self.oclIsTypeOf(Extension)) then {
                        topRels += map mapTopologicalRelationship(ffs, self.reference, currentStep);
                    } else {
                        previousDefaultStep := useCase.getPreviousStep(self.reference);
                        topRels += map mapTopologicalRelationship(ffs, previousDefaultStep, currentStep);
                    } endif;
                } endif;

                if (self.step->size() != stepNumber) then {
                    topRels += map mapTopologicalRelationship(ffs, currentStep, nextStep);
                } endif;
            };

            return topRels;
        }

        //Map current and next steps to a Topological Relationship
        mapping mapTopologicalRelationship(ffs : Set(TFM::FunctionalFeature),
```

```
230.                    sourceStep : UseCases::SingleEvent, targetStep : UseCases::Single
    Event) : TFM::TopologicalRelationship {
231.
232.            if (sourceStep.reference.oclIsUndefined() or sourceStep.reference = "") t
    hen {
233.                source := ffs->selectOne(ff | ff.id = sourceStep.id);
234.            } else {
235.                source := ffs->selectOne(ff | ff.id = sourceStep.reference);
236.            } endif;
237.
238.            if (targetStep.reference.oclIsUndefined() or targetStep.reference = "") t
    hen {
239.                target := ffs->selectOne(ff | ff.id = targetStep.id);
240.            } else {
241.                target := ffs->selectOne(ff | ff.id = targetStep.reference);
242.            } endif;
243.
244.            id := source.id + "," + target.id;
245.
246.            log("Creating Topological Relationship: "+id);
247.        }
248.
249.        //Get previous step of a given step
250.        helper UseCases::UseCase::getPreviousStep(currentStep : UseCases::DefaultEven
    t) : UseCases::DefaultEvent {
251.            var index : Integer;
252.            index := self.mainScenario.step->indexOf(currentStep);
253.            return self.mainScenario.step->at(index-1);
254.        }
255.
256.
257.        //Get merged pre-conditions
258.        helper UseCases::SingleEvent::getMergedPreConditions(referencedSteps : Set(Us
    eCases::SingleEvent)) : String {
259.            var condition : String;
260.            var useOr : Boolean := false;
261.
262.            condition := self.preconditions.getConditions();
263.
264.            if (condition.oclIsUndefined()) then {
265.                condition := "";
266.            }  endif;
267.
268.            if (referencedSteps->size() > 0) then {
269.                if (referencedSteps->hasPreConditions()) then {
270.
271.                    if (condition.size() > 0) then {
272.                        condition := condition+" OR ";
273.                    } endif;
274.
275.                    referencedSteps->forEach(step){
276.                        if (not step.preconditions.oclIsUndefined()) then {
277.                            if (useOr) then { condition := condition + " OR ";  } end
    if;
278.                            condition := condition+step.preconditions.getConditions()
    ;
279.                            useOr := true;
280.                        } endif;
281.                    };
282.
283.                } endif;
284.
285.            } endif;
286.
287.            if (condition.size()>0) then {
288.                condition := "("+condition+")";
289.            } endif;
290.
291.            return condition;
```

```
292.        }
293.
294.        //Get merged post-conditions
295.        helper UseCases::SingleEvent::getMergedPostConditions(referencedSteps : Set(U
    seCases::SingleEvent)) : String {
296.            var condition : String;
297.            var useOr : Boolean := false;
298.
299.            condition := self.postconditions.getConditions();
300.
301.            if (condition.oclIsUndefined()) then {
302.                condition := "";
303.            }  endif;
304.
305.            if (referencedSteps->size() > 0) then {
306.                if (referencedSteps->hasPostConditions()) then {
307.
308.                    if (condition.size() > 0) then {
309.                        condition := condition+" OR ";
310.                    } endif;
311.
312.                    referencedSteps->forEach(step){
313.                        if (not step.postconditions.oclIsUndefined()) then {
314.                            if (useOr) then { condition := condition + " OR ";  } end
    if;
315.                            condition := condition+step.postconditions.getConditions(
    );
316.                            useOr := true;
317.                        } endif;
318.                    };
319.
320.                } endif;
321.
322.            } endif;
323.
324.            if (condition.size()>0) then {
325.                condition := "("+condition+")";
326.            } endif;
327.
328.            return condition;
329.        }
330.
331.        //Check if preconditions exist
332.        helper Set(UseCases::SingleEvent)::hasPreConditions() : Boolean {
333.            var has : Boolean := false;
334.
335.            self->forEach (step) {
336.                if (not step.preconditions.oclIsUndefined()) then {
337.                    has := true;
338.                } endif;
339.            };
340.
341.            return has;
342.        }
343.
344.        //Check if postconditions exist
345.        helper Set(UseCases::SingleEvent)::hasPostConditions() : Boolean {
346.            var has : Boolean := false;
347.
348.            self->forEach (step) {
349.                if (not step.postconditions.oclIsUndefined()) then {
350.                    has := true;
351.                } endif;
352.            };
353.
354.            return has;
355.        }
356.
357.        //Recursively get the condition description
```

```
358.       helper UseCases::Condition::getConditions() : String {
359.           var cond : String;
360.           var size : Integer;
361.           var iteration : Integer = 0;
362.
363.           //SingleEvent or CompositeEvent
364.           if self.oclIsTypeOf(CompositeCondition) then {
365.               cond := "(";
366.               size := self.oclAsType(CompositeCondition).conditions->size();
367.
368.               self.oclAsType(CompositeCondition).conditions-
    >forEach (condition) {
369.                   iteration := iteration + 1;
370.                   cond := cond + condition.getConditions();
371.
372.                   if (iteration != size) then {
373.                       cond := cond + " " + self.oclAsType(CompositeCondition).opera
    tion.toString() + " ";
374.                   } endif;
375.               };
376.
377.               cond := cond + ")";
378.           } else {
379.               cond := self.oclAsType(SingleCondition).description;
380.           } endif;
381.
382.           return cond;
383.       }
```

Use Cases to TFM Transformation Parser Tools Java Class

```
1.  package lv.rtu.ldk.idm.usecases2tfm.blackbox;
2.
3.  import edu.stanford.nlp.ling.CoreLabel;
4.  import edu.stanford.nlp.ling.Label;
5.  import edu.stanford.nlp.objectbank.TokenizerFactory;
6.  import edu.stanford.nlp.parser.lexparser.LexicalizedParser;
7.  import edu.stanford.nlp.process.CoreLabelTokenFactory;
8.  import edu.stanford.nlp.process.PTBTokenizer;
9.  import edu.stanford.nlp.process.Tokenizer;
10. import edu.stanford.nlp.trees.Tree;
11. import java.io.StringReader;
12. import java.util.List;
13.
14. public class ParserTools
15. {
16.   private static final String NOUN_PHRASE = "NP";
17.   private static final String VERB_PHRASE = "VP";
18.   private static final String PERIOD = ".";
19.   private String grammar = "lib/englishPCFG.ser.gz";
20.   private String[] options = { "-maxLength", "80", "-retainTmpSubcategories" };
21.   private LexicalizedParser lp;
22.   private TokenizerFactory<CoreLabel> tokenizerFactory;
23.
24.   public ParserTools()
25.   {
26.     this.lp = new LexicalizedParser(this.grammar, this.options);
27.     this.tokenizerFactory = PTBTokenizer.factory(new CoreLabelTokenFactory(), "");
28.   }
29.
30.   public String getEntity(String sentence) {
31.     String result = "";
32.
33.     sentence = prepareSentence(sentence);
34.
```

```
35.     List rawWords = this.tokenizerFactory.getTokenizer(new StringReader(sentence)).t
    okenize();
36.     Tree parse = this.lp.apply(rawWords);
37.
38.     for (Tree tree : parse.children()[0].children())
39.     {
40.       if (tree.label().value().equalsIgnoreCase("NP")) {
41.
42.           //Check if the Noun Phrase consists of several words
43.           if (tree.getLeaves().size() == 1) {
44.               //Return the single word to be checked as the entity
45.               result = ((Tree)tree.getLeaves().get(0)).label().value();
46.           } else {
47.               //Return multiple words to be checked as the entity
48.               for (int i = 0; i < tree.getLeaves().size(); i++) {
49.                   result = result + ((Tree)tree.getLeaves().get(i)).label().value()
    + " ";
50.               }
51.               result = result.trim();
52.           }
53.
54.
55.
56.       }
57.     }
58.
59.     return result;
60.   }
61.
62.   public String getDescription(String sentence) {
63.     String result = "";
64.
65.     sentence = prepareSentence(sentence);
66.
67.     List rawWords = this.tokenizerFactory.getTokenizer(new StringReader(sentence)).t
    okenize();
68.     Tree parse = this.lp.apply(rawWords);
69.
70.     for (Tree tree : parse.children()[0].children()) {
71.       if (tree.label().value().equalsIgnoreCase("VP")) {
72.         result = resolveTree(tree).trim();
73.       }
74.     }
75.
76.     if (result.substring(0, 2).equalsIgnoreCase("is")) {
77.       result = result.substring(3);
78.     }
79.
80.     return result;
81.   }
82.
83.   private String resolveTree(Tree tree) {
84.     String result = "";
85.
86.     if (tree.children().length > 0) {
87.       for (Tree t : tree.children())
88.         result = result + resolveTree(t);
89.     }
90.     else {
91.       result = tree.label().value();
92.     }
93.
94.     return result.trim() + " ";
95.   }
96.
97.   public void analyzeString(String sentence)
98.   {
99.     sentence = prepareSentence(sentence);
100.
```

```
101.           List rawWords = this.tokenizerFactory.getTokenizer(new StringReader(sente
    nce)).tokenize();
102.           Tree parse = this.lp.apply(rawWords);
103.
104.           parse.pennPrint();
105.         }
106.
107.         private String prepareSentence(String sentence)
108.         {
109.           if (sentence.endsWith(".")) {
110.             return sentence;
111.           }
112.           return sentence + ".";
113.         }
114.       }
```

Use Cases to TFM Transformation Utilities Library Java Class

```
1.  package lv.rtu.ldk.idm.usecases2tfm.blackbox;
2.
3.  public class UtilitiesLibrary
4.  {
5.    private static ParserTools parserTools = new ParserTools();
6.
7.    public static String parseStepForEntity(String stepDesciption)
8.    {
9.      return parserTools.getEntity(stepDesciption);
10.   }
11.
12.   public static String parseStepForDescription(String stepDesciption) {
13.     return parserTools.getDescription(stepDesciption);
14.   }
15. }
```

Use Cases to TFM Transformation Run Java Class

```
1.  package lv.rtu.ldk.idm.usecases2tfm.part;
2.
3.  import java.io.IOException;
4.  import java.io.PrintWriter;
5.  import java.util.Collections;
6.  import java.util.List;
7.  import org.eclipse.core.resources.IFile;
8.  import org.eclipse.core.runtime.IPath;
9.  import org.eclipse.core.runtime.IStatus;
10. import org.eclipse.emf.common.util.BasicDiagnostic;
11. import org.eclipse.emf.common.util.EList;
12. import org.eclipse.emf.common.util.URI;
13. import org.eclipse.emf.ecore.resource.Resource;
14. import org.eclipse.emf.ecore.resource.ResourceSet;
15. import org.eclipse.emf.ecore.resource.impl.ResourceSetImpl;
16. import org.eclipse.jface.action.IAction;
17. import org.eclipse.jface.dialogs.MessageDialog;
18. import org.eclipse.jface.viewers.ISelection;
19. import org.eclipse.jface.viewers.IStructuredSelection;
20. import org.eclipse.m2m.qvt.oml.BasicModelExtent;
21. import org.eclipse.m2m.qvt.oml.ExecutionContextImpl;
22. import org.eclipse.m2m.qvt.oml.ExecutionDiagnostic;
23. import org.eclipse.m2m.qvt.oml.ModelExtent;
24. import org.eclipse.m2m.qvt.oml.TransformationExecutor;
25. import org.eclipse.m2m.qvt.oml.util.WriterLog;
```

```
26. import org.eclipse.swt.widgets.Shell;
27. import org.eclipse.ui.IObjectActionDelegate;
28. import org.eclipse.ui.IWorkbenchPart;
29. import org.eclipse.ui.IWorkbenchPartSite;
30. import org.eclipse.ui.console.ConsolePlugin;
31. import org.eclipse.ui.console.IConsole;
32. import org.eclipse.ui.console.IConsoleManager;
33. import org.eclipse.ui.console.MessageConsole;
34. import org.eclipse.ui.console.MessageConsoleStream;
35.
36. public class RunTransformationFileAction
37.   implements IObjectActionDelegate
38. {
39.   private static final String CONSOLE_NAME = "TRANSFORMATION_CONSOLE";
40.   private IWorkbenchPart targetPart;
41.   private URI domainModelURI;
42.
43.   private Shell getShell()
44.   {
45.     return this.targetPart.getSite().getShell();
46.   }
47.
48.   public void selectionChanged(IAction action, ISelection selection) {
49.     this.domainModelURI = null;
50.     action.setEnabled(false);
51.     if ((!(selection instanceof IStructuredSelection)) ||
52.       (selection.isEmpty())) {
53.       return;
54.     }
55.     IFile file = (IFile)((IStructuredSelection)selection)
56.       .getFirstElement();
57.
58.     IPath path = file.getFullPath();
59.
60.     this.domainModelURI = URI.createPlatformResourceURI(path.toString(), true);
61.     action.setEnabled(true);
62.   }
63.
64.   public void setActivePart(IAction action, IWorkbenchPart targetPart)
65.   {
66.     this.targetPart = targetPart;
67.   }
68.
69.   public void run(IAction action)
70.   {
71.     URI uri = URI.createPlatformPluginURI("lv.rtu.ldk.idm.usecases2tfm/lib/UseCases2
    TFM.qvto", true);
72.
73.     TransformationExecutor executor = new TransformationExecutor(uri);
74.
75.     ResourceSet resourceSet = new ResourceSetImpl();
76.     Resource inResource = resourceSet.getResource(this.domainModelURI, true);
77.     EList inObjects = inResource.getContents();
78.
79.     ModelExtent input = new BasicModelExtent(inObjects);
80.
81.     ModelExtent output = new BasicModelExtent();
82.
83.     ExecutionContextImpl context = new ExecutionContextImpl();
84.     context.setConfigProperty("keepModeling", Boolean.valueOf(true));
85.
86.     MessageConsole myConsole = findConsole("TRANSFORMATION_CONSOLE");
87.     MessageConsoleStream out = myConsole.newMessageStream();
88.     PrintWriter console = new PrintWriter(out);
89.     context.setLog(new WriterLog(console));
90.
91.     ExecutionDiagnostic result = executor.execute(context, new ModelExtent[] { input
    , output });
92.
```

```
93.     if (result.getSeverity() == 0)
94.     {
95.       List outObjects = output.getContents();
96.
97.       ResourceSet resourceSet2 = new ResourceSetImpl();
98.
99.       String resultModel = this.domainModelURI.toString();
100.              resultModel = resultModel.substring(0, resultModel.indexOf(".usecases")
    );
101.              resultModel = resultModel + ".tfm";
102.
103.              Resource outResource = resourceSet2.createResource(URI.createURI(result
    Model));
104.
105.              outResource.getContents().addAll(outObjects);
106.              try {
107.                outResource.save(Collections.emptyMap());
108.              } catch (IOException e) {
109.                MessageDialog.openError(getShell(), "Transformation failure", e.getMe
    ssage());
110.              }
111.
112.              console.flush();
113.           }
114.           else
115.           {
116.             IStatus status = BasicDiagnostic.toIStatus(result);
117.             String message = status.getMessage();
118.             MessageDialog.openError(getShell(), "Transformation failure", message);

119.           }
120.         }
121.
122.         private MessageConsole findConsole(String name)
123.         {
124.           ConsolePlugin plugin = ConsolePlugin.getDefault();
125.           IConsoleManager conMan = plugin.getConsoleManager();
126.           IConsole[] existing = conMan.getConsoles();
127.           for (int i = 0; i < existing.length; i++) {
128.             if (name.equals(existing[i].getName()))
129.               return (MessageConsole)existing[i];
130.           }
131.           MessageConsole myConsole = new MessageConsole(name, null);
132.           conMan.addConsoles(new IConsole[] { myConsole });
133.           return myConsole;
134.         }
135.       }
```

Use Cases to TFM Transformation Eclipse Plugin Manifest

```
1.  Manifest-Version: 1.0
2.  Bundle-ManifestVersion: 2
3.  Bundle-Name: %Bundle-Name
4.  Bundle-SymbolicName: lv.rtu.ldk.idm.usecases2tfm;singleton:=true
5.  Bundle-Version: 1.0.3
6.  Bundle-RequiredExecutionEnvironment: JavaSE-1.7
7.  Require-Bundle: org.eclipse.m2m.qvt.oml,
8.   lv.rtu.ldk.idm.tfm,
9.   lv.rtu.ldk.idm.usecases
10. Bundle-Vendor: %Bundle-Vendor
11. Bundle-ClassPath: lib/stanford-parser.jar,
12.  .,
13.  bin/
14. Import-Package: lv.rtu.ldk.idm.usecases2tfm.blackbox,
15.  lv.rtu.ldk.idm.usecases2tfm.part,
```

```
 org.eclipse.core.resources,
 org.eclipse.core.runtime;version="3.4.0",
 org.eclipse.jface.action,
 org.eclipse.jface.dialogs,
 org.eclipse.jface.viewers,
 org.eclipse.swt.widgets,
 org.eclipse.ui,
 org.eclipse.ui.console
Export-Package: edu.stanford.nlp.fsm,
 edu.stanford.nlp.international,
 edu.stanford.nlp.international.arabic,
 edu.stanford.nlp.international.arabic.pipeline,
 edu.stanford.nlp.international.arabic.process,
 edu.stanford.nlp.international.french,
 edu.stanford.nlp.international.french.pipeline,
 edu.stanford.nlp.international.morph,
 edu.stanford.nlp.io,
 edu.stanford.nlp.io.ui,
 edu.stanford.nlp.ling,
 edu.stanford.nlp.math,
 edu.stanford.nlp.objectbank,
 edu.stanford.nlp.parser,
 edu.stanford.nlp.parser.lexparser,
 edu.stanford.nlp.parser.lexparser.demo,
 edu.stanford.nlp.parser.metrics,
 edu.stanford.nlp.parser.tools,
 edu.stanford.nlp.parser.ui,
 edu.stanford.nlp.process,
 edu.stanford.nlp.process.treebank,
 edu.stanford.nlp.stats,
 edu.stanford.nlp.swing,
 edu.stanford.nlp.trees,
 edu.stanford.nlp.trees.international.arabic,
 edu.stanford.nlp.trees.international.french,
 edu.stanford.nlp.trees.international.hebrew,
 edu.stanford.nlp.trees.international.negra,
 edu.stanford.nlp.trees.international.pennchinese,
 edu.stanford.nlp.trees.international.tuebadz,
 edu.stanford.nlp.trees.tregex,
 edu.stanford.nlp.trees.tregex.tsurgeon,
 edu.stanford.nlp.util,
 edu.stanford.nlp.util.concurrent,
 lv.rtu.ldk.idm.usecases2tfm.blackbox,
 lv.rtu.ldk.idm.usecases2tfm.part
```

Use Cases to TFM Transformation Eclipse Plugin Configuration

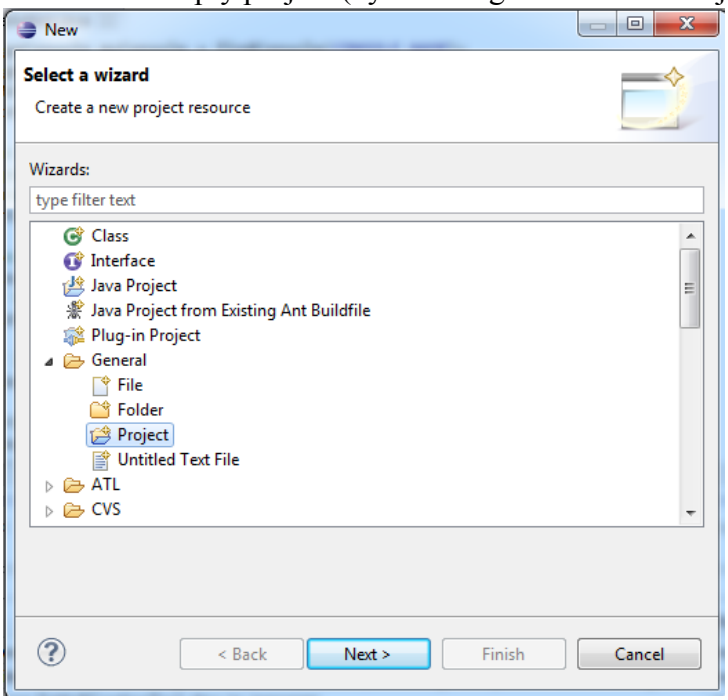
```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.2"?>
<plugin>

    <extension point="org.eclipse.m2m.qvt.oml.javaBlackboxUnits">
        <unit name="BlackBoxLibrary" namespace="lv.rtu.ldk.idm.usecases2tfm">
            <library name="UtilLib" class="lv.rtu.ldk.idm.usecases2tfm.blackbox.Util
itiesLibrary">
                <metamodel nsURI="http://lv/rtu/ldk/idm/usecases/1.0"/>
            </library>
        </unit>
    </extension>

    <extension point="org.eclipse.ui.popupMenus" id="%extension.id">
      <objectContribution
            id="lv.rtu.ldk.idm.usecases2tfm.part.RunTransformation"
            nameFilter="*.usecases"
            objectClass="org.eclipse.core.resources.IFile">
```

```
18.          <action
19.                label="%action.label"
20.                class="lv.rtu.ldk.idm.usecases2tfm.part.RunTransformationFileAction"
21.                menubarPath="additions"
22.                enablesFor="1"
23.                id="lv.rtu.ldk.idm.usecases2tfm.part.InitDiagramAction">
24.          </action>
25.       </objectContribution>
26.    </extension>
27.
28. </plugin>
```

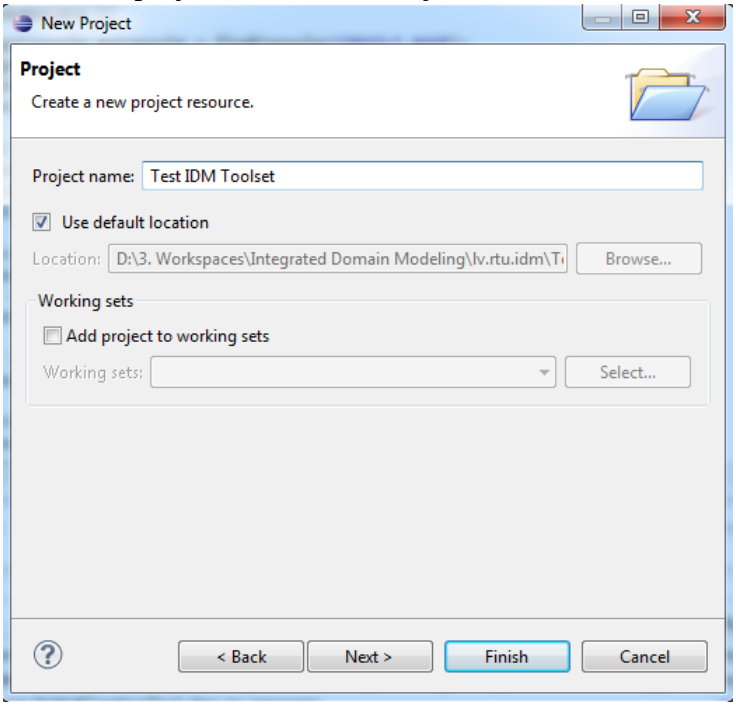
IDM TOOLSET USER GUIDE

Creating Use Cases with Use Cases Editor

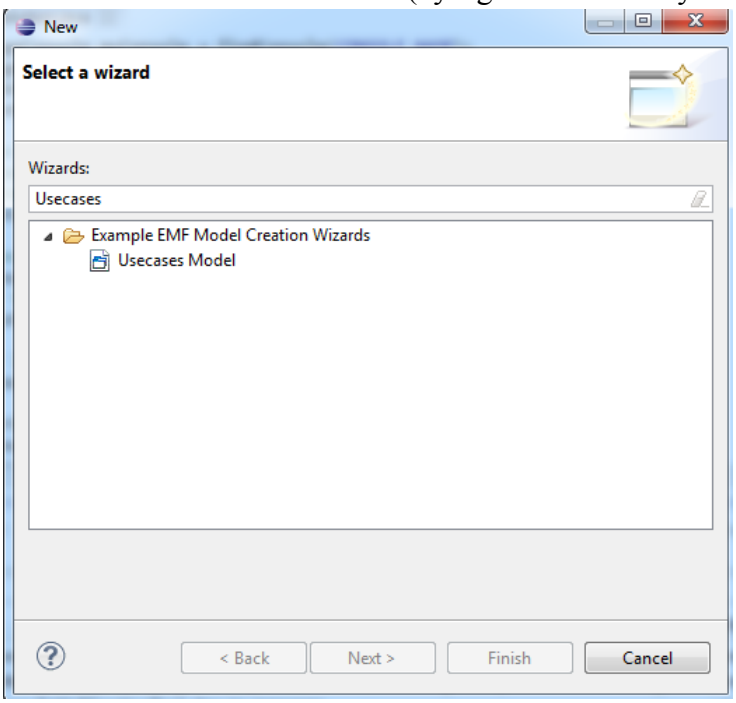
1. Launch Eclipse IDM
2. Create new empty project (by selecting File / New / Project…)



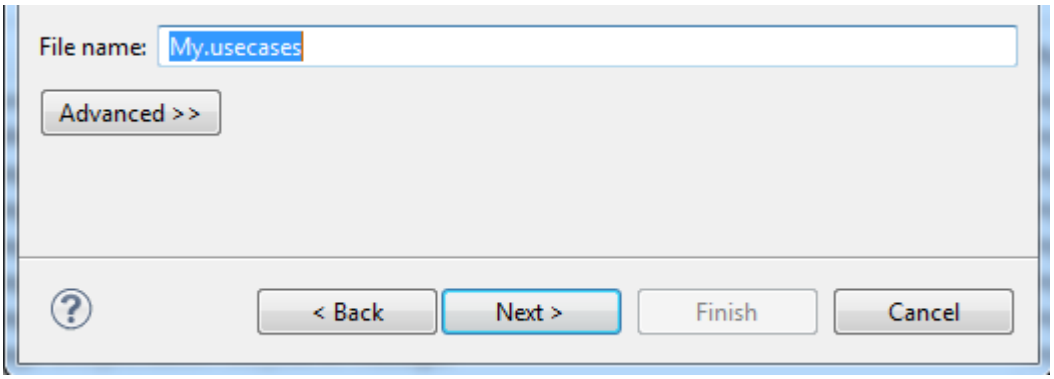
3. Name the project in the New Project wizard



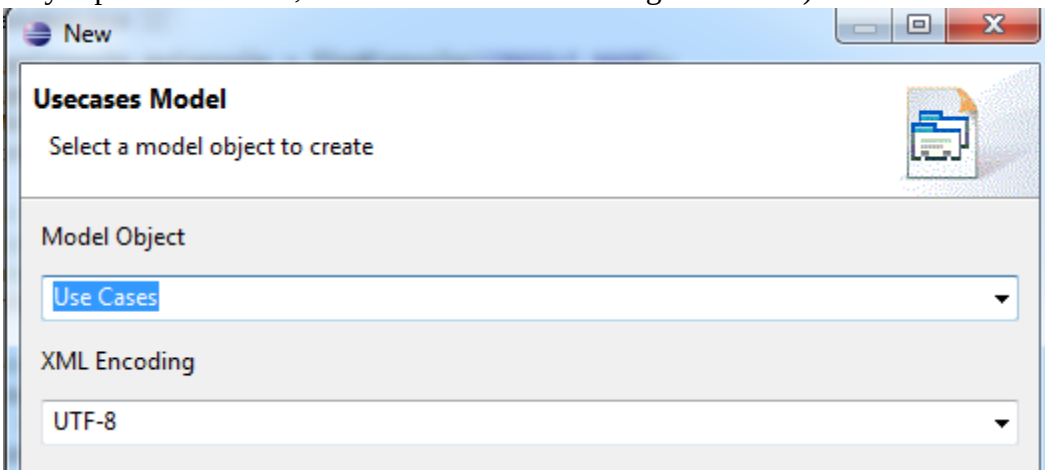
4. Create a new Use Cases model (by right click under your project New / Other…)



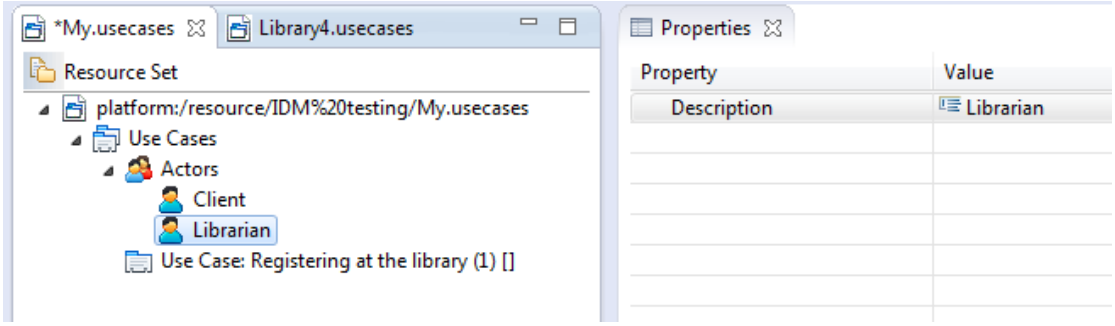
5. Name the Use Cases model in the New Use Cases wizard



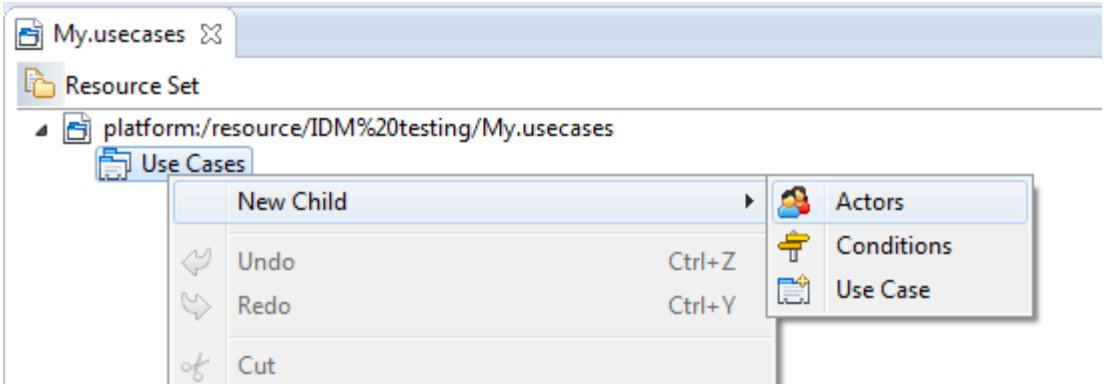
6. Choose the root of the Use Cases model to be “Use Cases” (it is possible to model also only separate elements, but this is not needed in a general case)



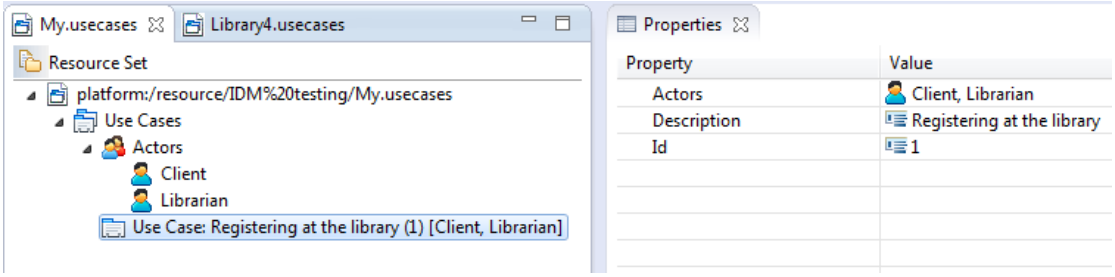
7. Create the Actors involved in the Use Cases (by right clicking on the Use Cases root element New Child / Actors)



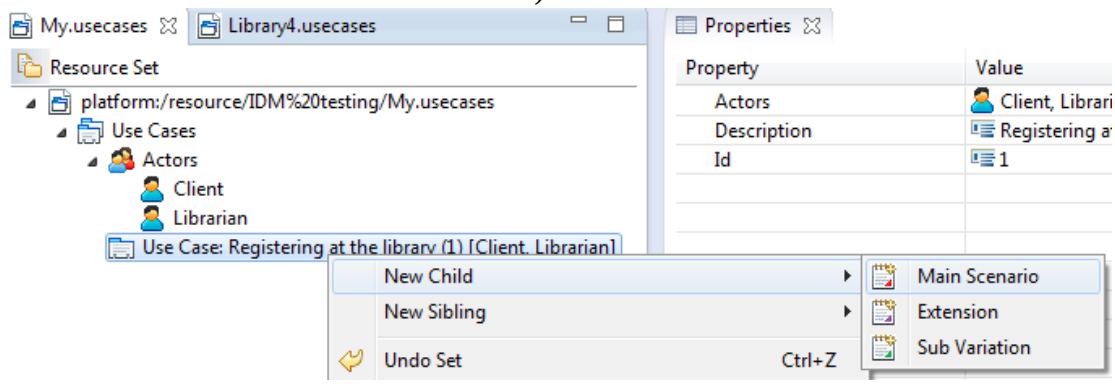
8. Start developing the Use Cases. Choose to add a Use Case node (by right clicking on the Use Cases root element New Child / Use Case)



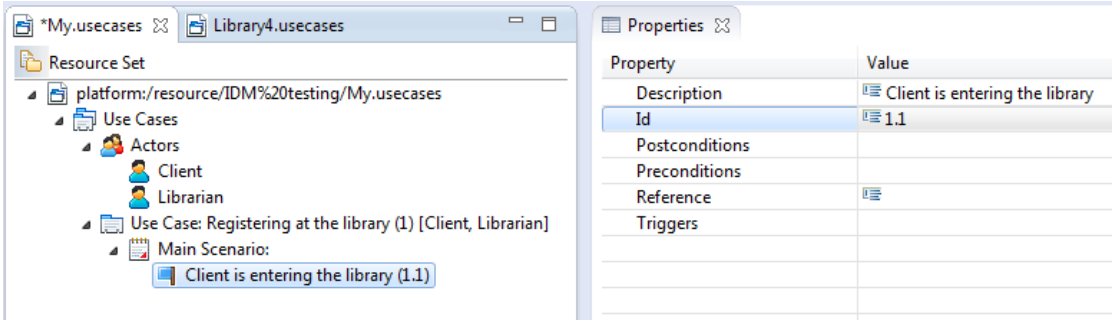
9. Define properties of the Use Case (by right clicking on the Use Case element and entering data in the properties pane)



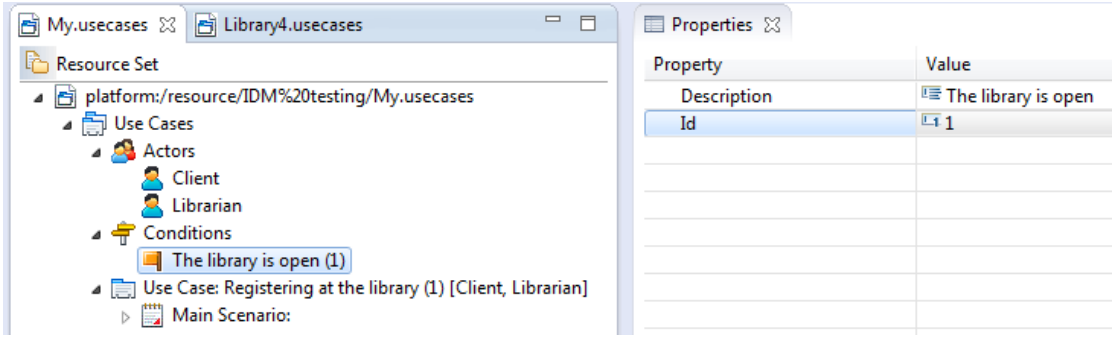
10. Continue developing Use Cases by creating Main Scenario (by right clicking on Use Case element New Child / Main Scenario)



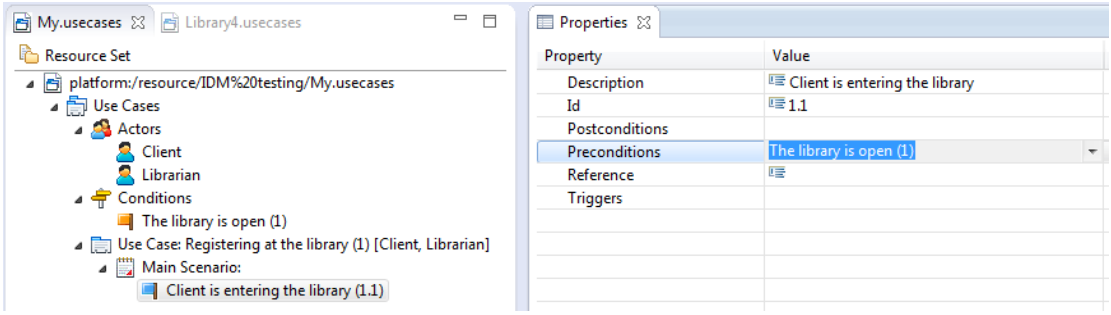
11. Create the events of the main scenario (by right clicking the Main Scenario element New Child / Default Step)



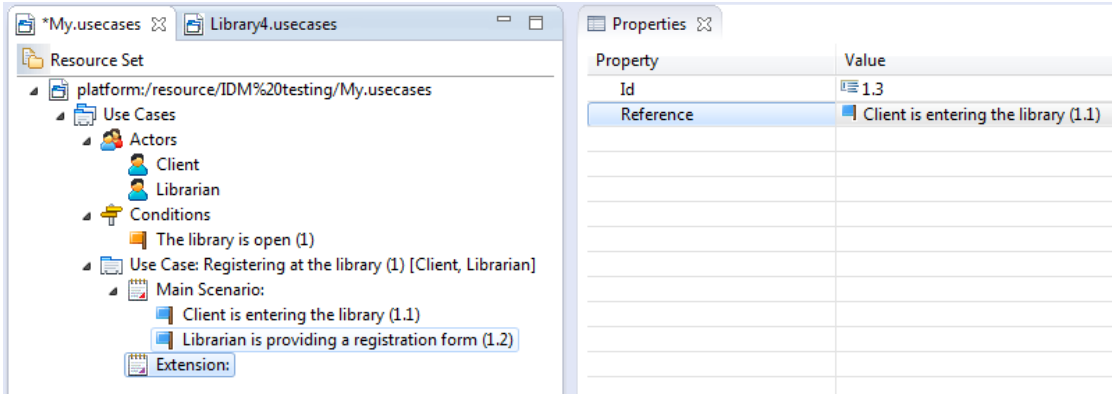
12. Define the conditions (by creating a Conditions elements right clicking it New Child / Condition)



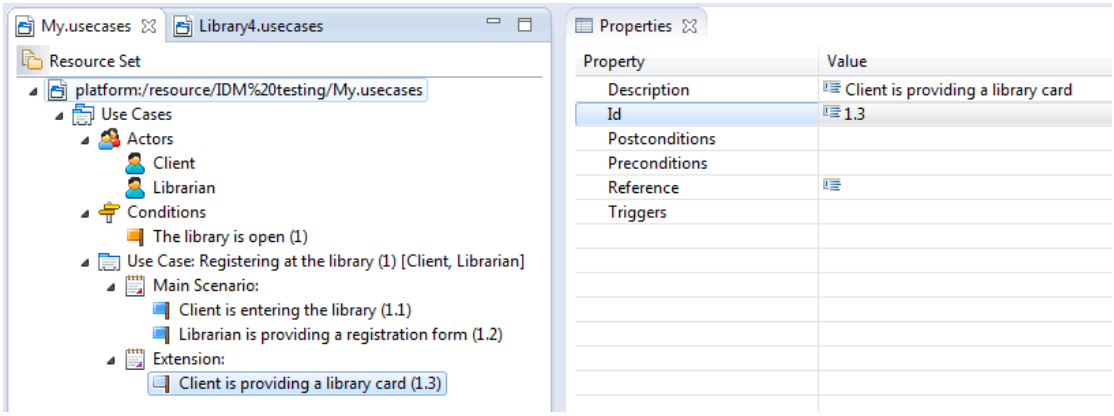
13. Assign the condition to a Use Case step in the properties pane



14. Create the Extension and Sub-variation scenarios under Use Case element

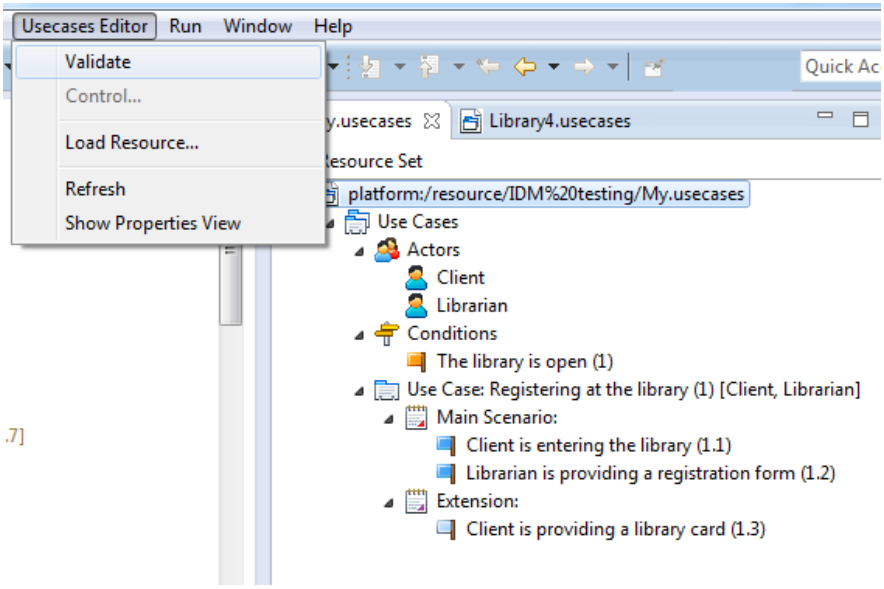


15. Create the alternative steps within the alternative scenarios

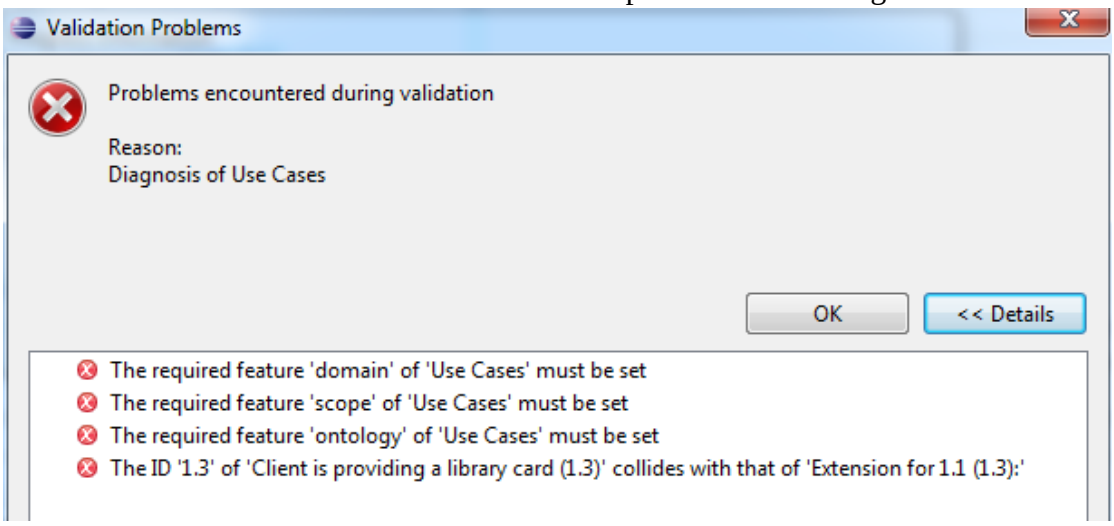


Validate the Use Cases model

1. Validate the Use Cases model (by selecting Usecases Editor / Validate), this performs the validation against the corresponding meta-model

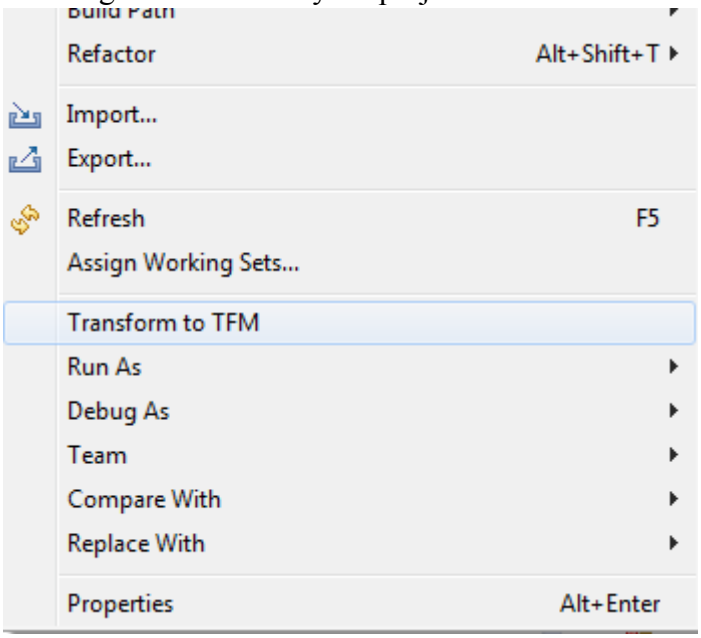


2. Correct the errors for the Use Cases elements as per the error messages under Details

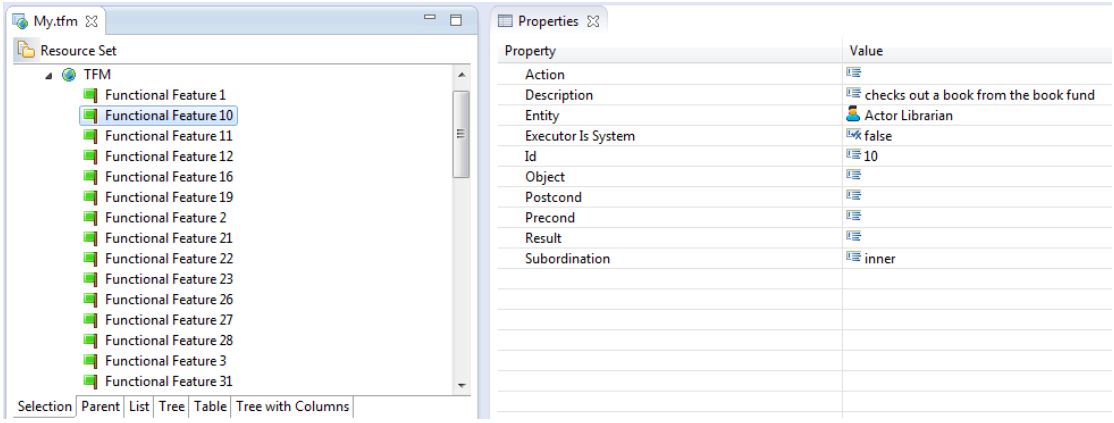


Generate the TFM model and diagram

1. Right-click the Use Cases model file and select “Transform to TFM”, a new model will be generated under your project with extension “.tfm”

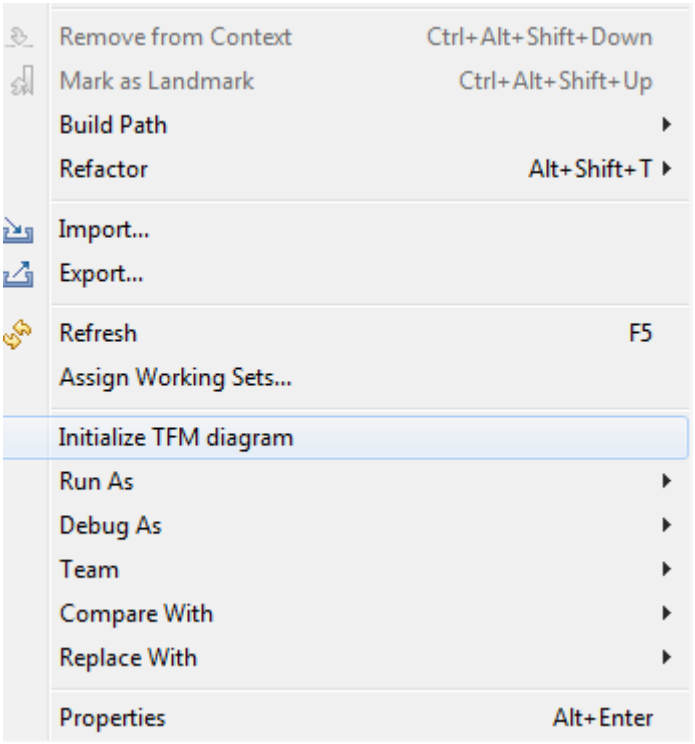


2. You can open the TFM model to see the results of the model transformation

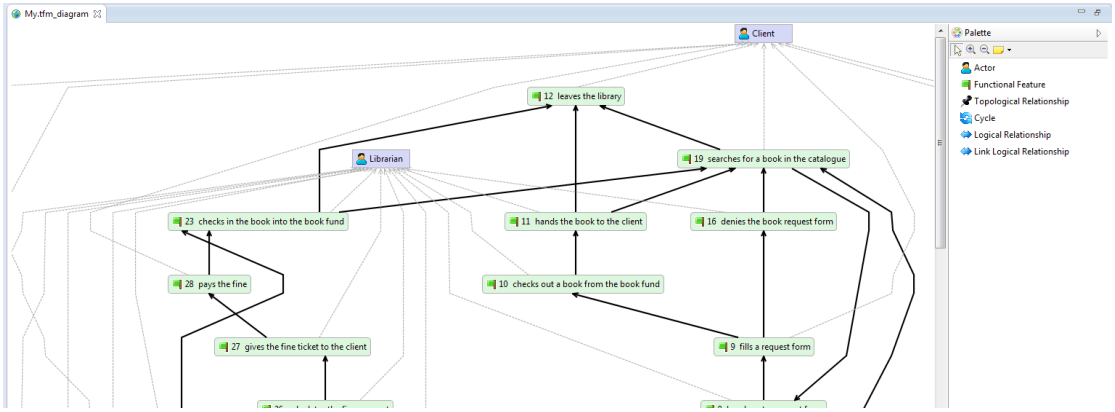


3. Right-click on the TFM model file and select “Initialize TFM diagram” to acquire a graphical representation. This will generate a new model with extension

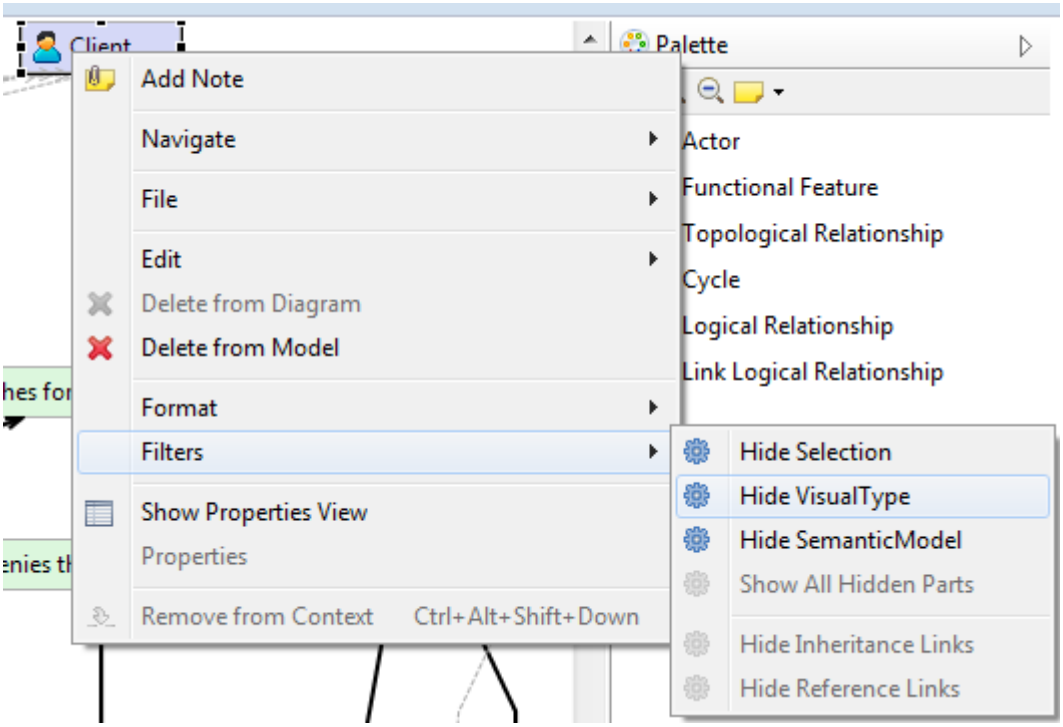
“.tfm_diagram”



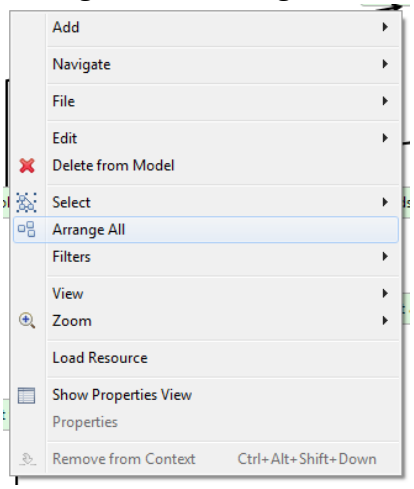
4. Open the generated TFM diagram file



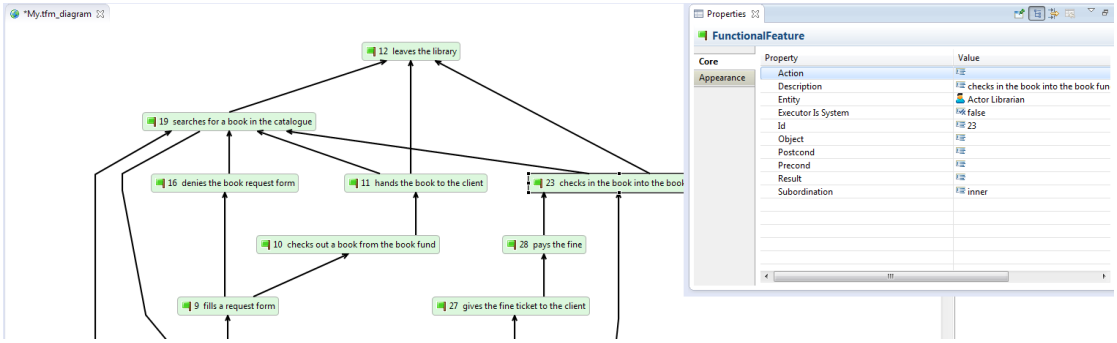
5. To acquire a classical TFM model view hide the Actor elements



6. Arrange the TFM diagram

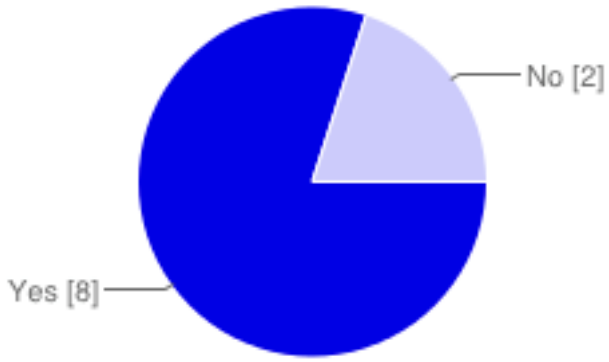


7. You have acquired a TFM model and a TFM diagram



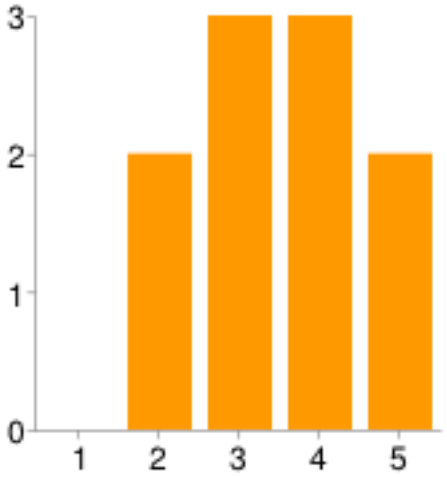
SURVEY ON THE IDM TOOLSET

Were you able to run the model transformation from Use Cases to TFM with your first try?



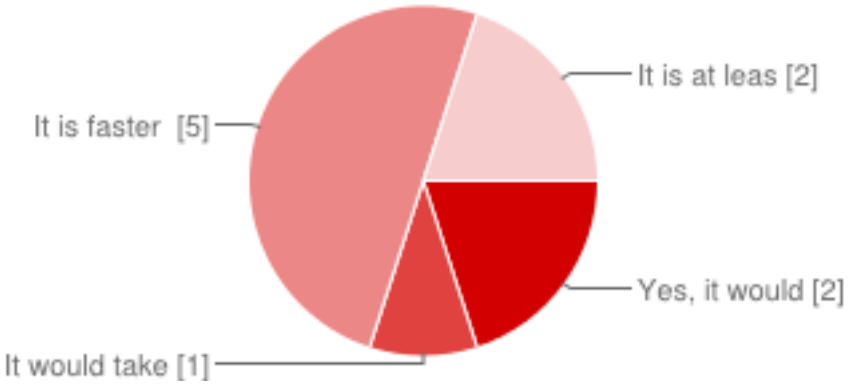
Yes	**8**	80%
No	**2**	20%

Please grade the understandability of this tool form 1 to 5



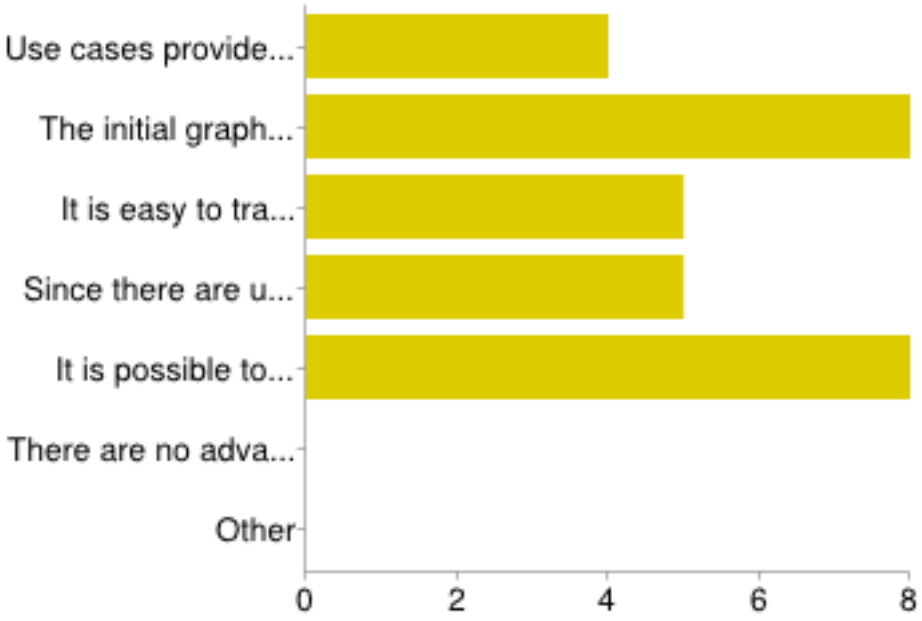
1	**0**	0%
2	**2**	20%
3	**3**	30%
4	**3**	30%
5	**2**	20%

If you had to construct a TFM manually without a tool (writing functional features in a table and drawing the graph), would it be faster for you than using the IDM toolset?



Yes, it would be faster without the IDM toolset	**2**	20%
It would take the same time	**1**	10%
It is faster to do it with IDM toolset	**5**	50%
It is at least 2 times faster to do it with IDM toolset	**2**	20%

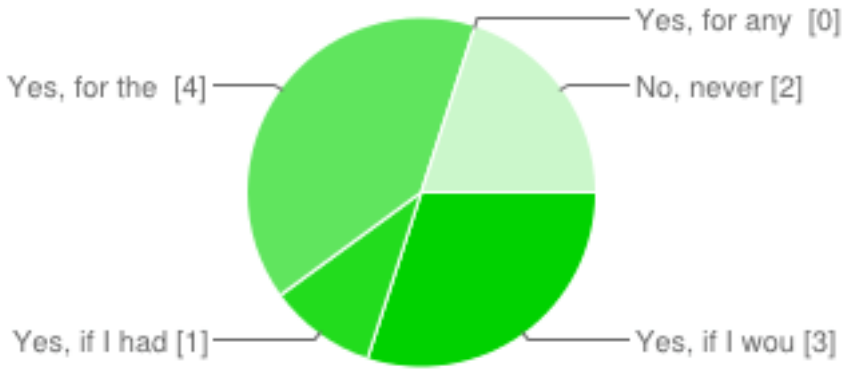
Please list the advantages of using the IDM toolset as oppose to constructing the TFM manually



Use cases provide a convenient way to capture procedural knowledge - business process	**4**	13 %

The initial graph is generated automatically from Use Cases	**8**	27 %
It is easy to trace back from functional features to Use Case steps	**5**	17 %
Since there are Use Cases it is easier validate the TFM, if it is correct and vice versa	**5**	17 %
It is possible to do mistake corrections in Use Cases and regenerate the TFM	**8**	27 %
There are no advantages	**0**	0%
Other	**0**	0%

Would you use the IDM toolset in your own project?



Yes, if I would need to acquire a graphical representation of the business process	**3**	30 %
Yes, if I had to capture my requirements with Use Cases (and getting TFM automatically is a bonus)	**1**	10 %
Yes, for the reasons above, but only for small projects	**4**	40 %
Yes, for any project	**0**	0%
No, never	**2**	20 %

Feedback

Toolset is easy to understand only after initial explanation/training. originally it is very confusing, because it is built around Eclipse. There are many options and buttons that are not used by the toolset that clutter the screen making it a lot harder to use without a tutorial. The input of Use cases can be misunderstood sometimes. The "Trigger" property of a scenario action that is vital for linking Use Cases can be

understood as both "this triggers it" and "it triggers this" and the only way to be sure which one it is, is by generating a TFM. I had to spend a lot of time re-doing all my triggers, just because I understood it the wrong way... A tooltip explaining what it means would resolve this. Manual Use Case numbering is also an issue. It can be useful sometimes, but an option for automatic numbering would make the tool a lot more user friendly Viens no trūkumiem varētu būt tas, ka specifiskas pogas (darbības) ir atrodamas izvēlnēs (ar peli jāklikšķina uz attiecīgā moduļa), būtu ērtāk ja pogas būtu augšējā navigācijā. Kā arī varētu veidot automātisku id palielināšanos, lai visu laik nav jāraksta manuāli tādas lietas. - To have a well-documented tutorial. - It requires longer time to re-accommodate the nodes in the graph than to insert the Use Cases. Some sort of automatic organization of the nodes on the graph could be handy.