Traceability and Interfacing Between Requirements Engineering and UML Domains using the Standardized ReqIF Format

Arne Noyer[1,2], Padma Iyenghar[1], Elke Pulvermueller[1], Florian Pramme[3] and Gert Bikker[3]
[1]*Institute for Software Engineering, University of Osnabrueck, Osnabrueck, Germany*
[2]*Willert Software Tools GmbH, Hannoversche Str. 21, Bueckeburg, Germany*
[3]*Institute for Distributed Systems, Ostfalia University, Salzdahlumer Str. 46/48, Wolfenbuettel, Germany*

Keywords: Requirements Interchange Format (ReqIF), Model-driven Development, Unified Modeling Language (UML), Requirements Engineering, Requirements Traceability.

Abstract: Model Driven Development (MDD) is deemed as a key to address the increasing complexity of software systems. It is imperative that the developed software fulfills the end-user's requirements. This implies that a collaboration between the Requirements Management (RM) tools and the modeling tools, enabling complete traceability and interfacing among these tools, is essential. On the other hand, existing tools collaborating between RM and modeling tools support a very limited sub-set of new features (e.g. traceability analysis) and are compatible with only a few tools. As a result, software engineers are often required to educate themselves on another (often complex), intermediate (collaborating) tool, merely to realize a very limited sub-set of supported features. This paper addresses these gaps and introduces an approach for exchanging information between RM tools and Unified Modeling Language (UML) tools by using the standardized Requirements Interchange Format (ReqIF). The proposed approach (a) enables software developers to create links between requirements and UML elements in their modeling tool and (b) facilitates requirements engineers to make traceability/other analyses down to linked model elements inside their RM tool. In contrast to many other approaches, no additional user interface is needed for traceability.

1 INTRODUCTION

In the series of evolution in software engineering, Model Driven Architecture (MDA) can be considered as the next/ongoing paradigm shift. Towards this direction, in the recent decade, MDD (France et al., 2006) has already made significant inroads for its application in software engineering projects. Some examples of tools used for MDD in software engineering projects are Rhapsody (IBM, 2014b), Enterprise Architect (Sparx Systems, 2014), Papyrus (Eclipse Foundation, 2013a) and Matlab/Simulink (Mathworks, 2014). In this paper, the focus is on the Unified Modeling Language (UML) for MDD. The UML (Object Management Group, 2013b) is one among the widely used industry standards for MDD (France et al., 2006). Besides general UML elements, UML profiles enable to address application areas such as real-time embedded systems and system engineering.

While the software requirements engineering process by itself may vary widely on the application domain, it is imperative that the requirements specified for the software are met during system modeling and development (e.g. using a UML tool). This implies that, while collaborating between a RM tool and a modeling tool, there needs to be complete traceability and interfacing between the requirements engineering tool and the MDD tool. This aspect gains further significance during the development of safety-certified software.

However, UML tools (e.g. Rhapsody) have to interface with further interfacing tool(s) (e.g. Rational Rhapsody Gateway (IBM, 2010)) that may provide an interface between the UML tool and RM tools. This is required, for example, to create links between requirements and model elements and to provide additional analysis capabilities to support traceability and requirements analysis. However, many interfacing tools only allow to do such things inside their own user interfaces. Thereby, engineers have to learn how to work with another tool, which can be quite com-

Noyer A., Iyenghar P., Pulvermueller E., Pramme F. and Bikker G..
Traceability and Interfacing Between Requirements Engineering and UML Domains using the Standardized ReqIF Format.
DOI: 10.5220/0005327303700375
In *Proceedings of the 3rd International Conference on Model-Driven Engineering and Software Development* (MODELSWARD-2015), pages 370-375
ISBN: 978-989-758-083-3

plex. This can result in a lower acceptance rate for a traceability solution. Further, most of the interfacing tools are proprietary for one modeling tool and only support data exchange with certain RM tools.

Addressing the aforementioned gaps this paper presents a framework for complete traceability and interfacing between software requirements engineering tools and modeling tools. It presents the following novel contributions with initial prototype implementation and results.

- Usage of the standardized ReqIF format to transfer representations of requirements to UML tools
- Analysis of elements in UML tools, which are linked to requirements
- Creation of representations for linked model elements inside the ReqIF file
- Enables traceability and impact analyses directly in UML and RM tools

2 STATE OF THE ART AND RELATED WORK

In requirements engineering, traceability is very important. Often, there are different refinement levels of requirements (e.g. user requirements and system specification) in RM tools. Requirements of these levels are linked to each other in order to ensure consistency. If a requirement changes, a requirements engineer can check whether linked requirements have to be changed accordingly or analyze the impact. There are already powerful mechanisms for such a traceability and other analyzes in many RM tools (e.g. (IBM, 2014a) and (Polarion Software, 2014)).

In the requirements management domain, the Requirements Interchange Format (ReqIF) (Object Management Group, 2013a) has become a standard by the Object Management Group (OMG) (Object Managemeng Group (OMG), 2014) for exchanging requirements between different RM tools. It is a normative XML-based data format to exchange and store requirement information between different tools and tool chains. ReqIF is an ensuing descendant of the RIF standard defined by Hersteller Initiative Software (HIS) (Hersteller Initiative Software (HIS), 2014) working group in the automotive domain.

ReqIF even allows that requirements of different levels are managed in different tools. For instance, the user requirements could be managed by a client in his RM tool. He can export them into a ReqIF file and then send them to a supplier. On the other hand, the supplier can import the requirements into another RM tool, create system specification requirements in this tool and create links to the user requirements. Afterwards, the system specification requirements and their links to user requirements can be exported in a ReqIF file and can be send back to the client. The client can import the ReqIF file again and has complete traceability information in his RM tool. Nowadays, many RM tools support the ReqIF format for import/export.

Unfortunately, for exchanging requirements and/or traceability information with MDD, there is not such a standard. Most of the solutions, which are available on the market, are proprietary for a certain modeling tool, as shown in Fig. 1. They are based on (third party) interfacing tools, which are only compatible with a selection of the RM tools and other sources for requirements (e.g. Rational Rhapsody Gateway (IBM, 2010)).

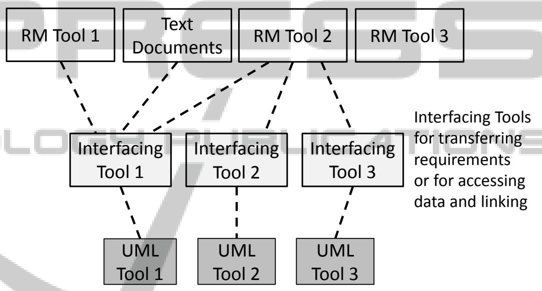


Figure 1: Proprietary Interfacing Tools for Different UML Tools.

There is a very detailed survey about traceability in RM and MDD in (Winkler and von Pilgrim, 2010). Among other topics, traceability in practice, overcoming (current) limitations and future challenges are discussed. As practice, it is mentioned that requirements traceability is usually managed in RM tools. When a traceability to other tools is needed, success stories are about companies, which developed their own proprietary traceability tools for their internal usage.

As technical limitations, it is mentioned in (Winkler and von Pilgrim, 2010), that creating and maintaining traceability links is not supported enough by tools. Further, there is a lack of integration between different tools and quite some effort has to be put in keeping the links up to date.

The survey (Winkler and von Pilgrim, 2010) analyzes two categories for overcoming integration issues: integrative approaches and specialized traceability tools. The specialized tools do all have their own editors and user interfaces, which makes them less intuitive, but traceability activities are supported very well. Some even allow automatic trace recording.

Integrative approaches, discussed in (Winkler and von Pilgrim, 2010), should be compatible with existing RM and MDD tools. Of course, the trace information has to be stored in some way. In (Walderhaug et al., 2008) an approach is proposed, which uses a central repository for traceability information. Other tools have to be integrated to this solution by using its API. All the integrative approaches do not manage the traceability links in the original tools, in which the linked artifacts were created. This implies, that the traceability links can not be viewed and analyzed in the original tools. Further, it is pointed out in (Winkler and von Pilgrim, 2010) (page 548): ”Users that are already familiar with their development tools usually do not want to switch tools just because some other tool supports traceability”. Therefore, the goal of the proposed approach is that users can do everything in their tools.

In recent research, there is another integrative approach described in (Graf and Jastram, 2011). Components of the prototype implementation seem to be part of the product Yakindu Traceability (itemis AG, 2014) now. The approach provides its own traceability meta-model and allows an integration with different tools and formats by using so called Trace Point Providers, which are basically like plug-ins. There is already a Trace Point Provider for the ReqIF format, which implies that it is compatible with a wide variety of RM tools. Further, there is a Trace Point Provider for UML models from Papyrus (Eclipse Foundation, 2013a). Papyrus uses the Eclipse-based implementation (Eclipse Foundation, 2013b) of the standardized UML meta-model. Hence, it is possible to have a traceability between requirements in ReqIF files and UML model elements from Papyrus. Additionally, Yakindu Traceability allows to define link types and which kinds of artifacts they can have as source and target. Furthermore, it has a powerful Eclipse-based user interface for creating links, their visualization and for traceability analyzes. While this approach seems to be flexible and extensible to be compatible with different tools, the same challenges as for other integrative approaches apply.

Open Services for Lifecycle Collaboration (OSLC) (Open Services for Lifecycle Collaboration (OSLC), 2014) is not only a technology for integrating RM tools with MDD tools, but a general approach for exchanging data between different tools (Ejaz Ul Haq, 2013). It is already used for an integration between Rhapsody (IBM, 2014b) and Doors (IBM, 2014a) by IBM. Creating OSLC interfaces for tools enables them to directly access each others data. Thus, tools have to implement which data they want to visualize and what users are able to do with them. Further, currently only a few RM and MDD tools support OSLC. Furthermore, the IT infrastructure must allow such a communication between tools. This may not be always available; for instance, in a situation when the requirements engineer is an employee from one company and has to send the specified requirements to a software developer in another company.

3 INTERFACING BETWEEN RM AND MDD TOOLS WITH REQIF

Since the ReqIF format is a standard for exchanging requirements and traceability is already well established in RM, ReqIF is also especially suitable for exchanging requirements with a modeling tool. A proposed workflow for this exchange is illustrated in Fig 2.

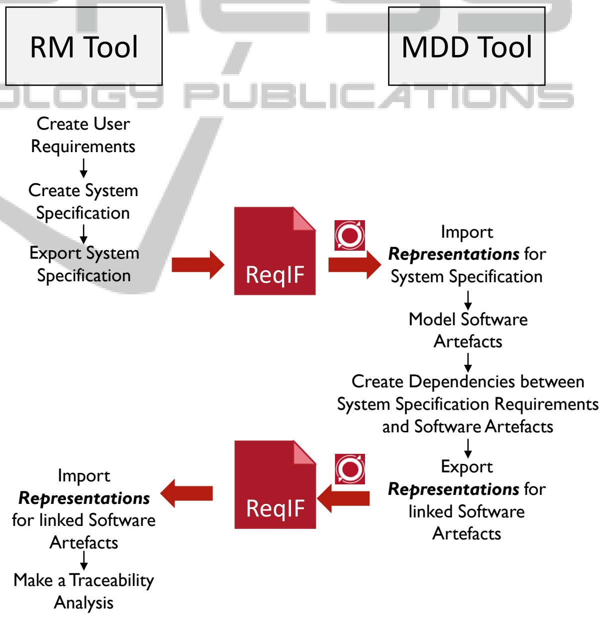

RM Tool
MDD Tool
Create User Requirements
Create System Specification
Export System Specification
ReqIF
Import *Representations* for System Specification
Model Software Artefacts
Create Dependencies between System Specification Requirements and Software Artefacts
Export *Representations* for linked Software Artefacts
ReqIF
Import *Representations* for linked Software Artefacts
Make a Traceability Analysis

Figure 2: Proposed workflow for working with reqirements in MDD.

On the left side, it is depicted what a requirements engineer creates in his RM tool, while on the right side, the work of a software developer in a MDD tool is depicted. For a high acceptance rate of the approach, both have to be able to do all their work in their established RM and MDD tool. A requirements engineer exports the requirements, which are relevant for the software developer, into a ReqIF file. Then, the ReqIF file is sent to the software developer. The software developer in turn imports representations of the requirements into his MDD tool. Since not every MDD tool necessarily supports requirements natively, they may have to be represented by other elements, depending on the MDD tool. Afterwards, model artifacts can be linked with requirements by using native mechanisms of the MDD tool (e.g. by using UML Dependencies). In order to transfer the traceability information back to the RM tool, representations of linked model artifacts are created in the ReqIF file and are also linked to the requirements in it. Finally, those representations of linked model artifacts can be imported into the RM tool. In practice, the requirements should only be changed inside the RM tool and the model artifacts only inside the MDD tool to avoid redundancy issues. Thereby, loss of data can be avoided, when changes inside representations for elements of the other domain are made. The workflow is similar to what business partners already do when exchanging requirements of different refinement levels, but extended to the domain of MDD.

The proposed workflow (Fig. 2) enables requirements engineers to make a traceability analysis and a coverage analysis inside the RM tool. In order to allow software developers to make an impact analysis inside the MDD tool, representations of changed requirements have to be marked as changed (e.g. by using a UML Stereotype) when updating requirements. Then, the software developer can check if linked model artifacts have to be modified. Any analysis can be made inside the already used tools. This also reduces the amount of needed licenses for tools. A software developer, for instance, does not need to have a RM tool installed on his computer.

In the proposed approach, the representations of model artifacts are stored in the ReqIF format; just like requirements as SpecObject (Object Management Group, 2013a) elements of the ReqIF format. SpecObjects reference a certain SpecType to indicate what kind of requirements they are. Since the ReqIF format allows to define attributes for a SpecType, information about the model artifacts, such as their name and type, are stored inside the ReqIF file. Values of these attributes can then be shown for their representations inside the RM tool.

Requirements, on the other hand, have to be represented inside the MDD tools. For the work discussed in this paper, the focus for MDD tools is on the usage of UML tools. Some of them, such as Rhapsody (IBM, 2014b), natively support the creation of requirements in their models. In such tools it is obvious to use them to represent requirements from RM tools. However, the UML standard itself does not define elements for requirements. Therefore, not every UML tool has them, but the UML can be extended by using profiles. The Systems Modeling Language (SysML) (Object Managemeng Group (OMG), 2012) is a profile for the UML, which also contains support for requirements. Thus, it is possible to apply a profile, such as the SysML, to a UML model to enable it to work with requirements.

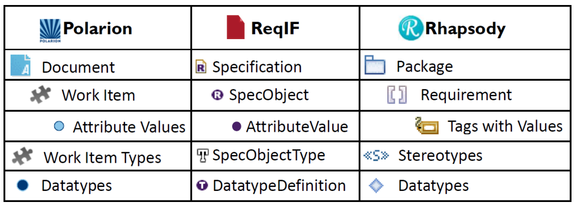
There are different possibilities for interfacing with a UML tool. Most of them allow the export/import model elements from a XMI format, which contains a standardized UML model. Therefore, it seems like a suitable way to create a ReqIF to (standardized) UML interface. Thereby, the representations of requirements would be created inside the standardized UML format and then transferred to different UML tools. Unfortunately, as stated in (Noyer et al., 2014), there are differences in how UML tools use the standardized UML format for export/import. It would have to be structured differently for different tools. Furthermore, the usage of UML profiles for creating requirements would be needed, since UML does not support requirements natively. Then, when transferring the requirements to a concrete UML tool, they may not be mapped to their proprietary requirement elements, as they exist for some tools such as Rhapsody.

Most UML tools also allow to access their models by using Application Programming Interfaces (API). They can directly be used for implementing ReqIF interfaces to different UML tools. The disadvantage is that this leads to proprietary solutions for every UML tool. On the other hand, there is the advantage that proprietary requirements elements from different UML tools can be considered. In order to realize this by using the standardized UML format, the tool specific behavior has to be considered anyway.

4 INITIAL EXPERIMENTAL EVALUATION

A prototype of the proposed mechanism for directly interfacing between ReqIF files and the UML tools IBM Rational Rhapsody and Enterprise Architect, which enables the usage of the workflow (as shown in Fig. 2) has been implemented. The direct integration with these tools was made to be able to use their proprietary model elements for requirements.

The ReqIF file is parsed by using the Requirements Modeling Framework (RMF) (Eclipse Foundation, 2014) and the elements in the UML tools are created/analyzed by using their Java APIs. This allows to make a traceability analysis inside a RM tool and an impact analysis inside the UML tools. The prototype has been evaluated by exporting requirements from a RM tool, Polarion (Polarion Software, 2014) via ReqIF and transferring representations of them to Rhapsody and Enterprise Architect. In future work, an integration with the standardized UML XMI format is planned to support a wide variety of UML tools. Thereby, the SysML profile is applied for creating requirements in UML.



Polarion	ReqIF	Rhapsody
Document	Specification	Package
Work Item	SpecObject	Requirement
Attribute Values	AttributeValue	Tags with Values
Work Item Types	SpecObjectType	Stereotypes
Datatypes	DatatypeDefinition	Datatypes

Figure 3: Mapping from requirements in Polarion and ReqIF to their representations in Rhapsody.

Figure 3 visualizes an extract of how requirements are exported from Polarion into a ReqIF file and how the prototype creates representations for them in the UML tool Rhapsody. For each document with requirements (in Polarion they are called Work Items), a package is created inside Rhapsody. In these packages, there is a representation for every requirement of the corresponding document. Since Rhapsody does provide model elements for requirements, these are used. The requirements are structured hierarchically, just as they are structured in Polarion and ReqIF (Fig. 5). Requirements do further have types (e.g. functional requirement) with attribute definitions, such as author or creation time. In the UML domain, these are mapped to Stereotypes with Tags. Therefore, every requirement representation in Rhapsody has a stereotype assigned and its attribute values (e.g. creation time) are stored inside tagged values. In Rhapsody, these representations of requirements can be linked with other UML model elements by using UML Dependencies (Fig. 4).

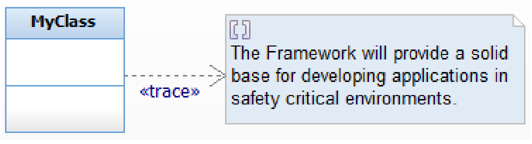
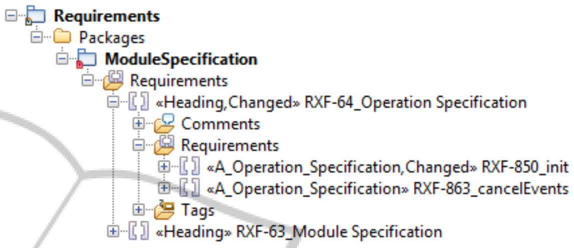

MyClass
«trace»
The Framework will provide a solid base for developing applications in safety critical environments.

Figure 4: A class inside Rhapsody, which is linked to a representation of a requirement from Polarion.

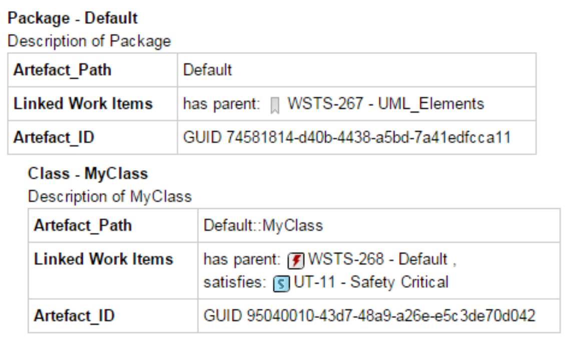
When the requirements are changed during development and updated requirements are re-transfered to Rhapsody, the prototype compares the already existing requirements representations with the updated requirements. In most cases, it should be enough to compare the attribute *lastChanged*, which every SpecObject (requirement) has inside ReqIF files. However, to ensure a correct comparison, the prototype also allows to compare all attributes. If the prototype identifies a changed requirement, it assigns a stereotype Changed to it (Fig 5)). Further, if there is a representation of a requirement, the corresponding document is inside the updated ReqIF file and there the requirement does not exist anymore, a stereotype *Deleted* is assigned to the requirement representation in Rhapsody. Since manually searching for elements is a tedious task, Rhapsody allows to search for model elements by defining search queries. Hence, a complete impact analysis can be made inside Rhapsody.



```
Requirements
  Packages
    ModuleSpecification
      Requirements
        «Heading,Changed» RXF-64_Operation Specification
          Comments
          Requirements
            «A_Operation_Specification,Changed» RXF-850_init
            «A_Operation_Specification» RXF-863_cancelEvents
          Tags
        «Heading» RXF-63_Module Specification
```

Figure 5: Changed requirements in Rhapsody.

Further, an algorithm was implemented for searching the model elements inside Rhapsody and Enterprise Architect, which are linked to requirements with UML Dependencies, and to create representations for them inside the ReqIF file. Therefore, at first a Specification (document) and a SpecObjectType (compare Fig. 3) are created inside the ReqIF file. Afterwards, SpecObjects are created for each linked UML model element inside the Specification and the SpecObjectType is set. Further, attributes are created for storing information for the model from the UML tool like its type (e.g. UML Class), description, identifier and path inside the package structure. Additionally, a link to the requirement inside the ReqIF file is created accordingly to the Dependency inside the UML tool. The ReqIF file can be re-imported into the used RM tool now.



Package - Default
Description of Package

Artefact_Path	Default
Linked Work Items	has parent: WSTS-267 - UML_Elements
Artefact_ID	GUID 74581814-d40b-4438-a5bd-7a41edfcca11

Class - MyClass
Description of MyClass

Artefact_Path	Default::MyClass
Linked Work Items	has parent: WSTS-268 - Default , satisfies: UT-11 - Safety Critical
Artefact_ID	GUID 95040010-43d7-48a9-a26e-e5c3de70d042

Figure 6: UML elements in Polarion

Figure 6 shows how representations of a UML Package and a UML Class are visualized in a document of the RM tool Polarion. The complete traceability information is inside the RM tool and can be used for different kinds of analyses, for which modern day RM tools already provide powerful mechanisms. Furthermore, the information can be considered during document generation from RM tools, as it may be needed for certification projects. The prototype implementation is currently evaluated in a certification project for this purpose. Please note that additional implementation details of this prototype are available at (Willert Software Tools, 2014).

5 CONCLUSION AND FUTURE WORK

The paper discusses an approach for using the standardized ReqIF format to integrate the UML domain and the RM domain. A prototype implementation of this approach interfaces between ReqIF and the UML tools Rhapsody (IBM, 2014b) and Enterprise Architect (Sparx Systems, 2014) by using their Java APIs. This allows a requirements engineer to make a traceability and coverage analysis directly inside his RM tool and a software developer to make an impact analysis inside his UML tool. In order to increase the acceptance rate of the approach, both can work in the tools they are used to, and they do not have switch to another tool just because of traceability.

Some future directions are: to support the standardized UML XMI format, to support the open-source UML tool Papyrus (Eclipse Foundation, 2013a), to roundtrip UML diagrams to RM tools, to support Dependencies in UML tools with custom Stereotypes, to enable automatically invoking the interfacing between ReqIF and UML tools (e.g. on a build server), the realization of a OSLC interface, and to investigate how the approach can be adapted to other non-UML MDD tools (e.g. Matlab/Simulink (Mathworks, 2014)).

REFERENCES

Eclipse Foundation (2013a). Papyrus UML Product website. http://www.eclipse.org/papyrus/.

Eclipse Foundation (2013b). UML2 Project website. http://www.eclipse.org/modeling/mdt/?project=uml2.

Eclipse Foundation (2014). Requirements Management for Eclipse - Requirements Modeling Framework (RMF). http://www.eclipse.org/rmf/.

Ejaz Ul Haq (2013). Tool integration using OSLC. Master Thesis, Mlardalen University, Vsteras, Sweden, http://www.idt.mdh.se/utbildning/exjobb/files/ TR1420.pdf.

France, R. B., Ghosh, S., Dinh-Trong, T., and Solberg, A. (2006). Model-driven development using UML 2.0: promises and pitfalls. *Computer*, 39:59 – 66.

Graf, A. and Jastram, M. (2011). Requirements, Traceability and DSLs in Eclipse with the Requirements Interchange Format (RIF/ReqIF). http://deploy-eprints.ecs.soton.ac.uk/307/1/mbees2011.pdf.

Hersteller Initiative Software (HIS) (2014). HIS Website. http://www.automotive-his.de.

IBM (2010). Rational Rhapsody Gateway Add On User Manual. http://pic.dhe.ibm.com/infocenter/rhaphlp/v7r5/topic/com.ibm.rhapsody.oem.pdf.doc/pdf/manual.pdf.

IBM (2014a). Rational DOORS Website. http://www-03.ibm.com/software/products/en/ratidoor.

IBM (2014b). Rational Rhapsody Family Website. http://www-03.ibm.com/software/products/en/ratirhapfami.

itemis AG (2014). Yakindu Traceability - Trace Management and Reporting (Product Website). http://www.yakindu.com/traceability/.

Mathworks (2014). Matlab/Simulink Website. http://www.mathworks.co.uk/products/simulink/index.html.

Noyer, A., Iyenghar, P., Pulvermueller, E., Pramme, F., Engelhardt, J., Samson, B., and Bikker, G. (2014). Tool Independent Code Generation for the UML - Closing the Gap between Proprietary Models and the Standardized UML Model. In *Evalutation to Novel Approaches of Software Engineering (ENASE 2014)*, Lisbon, Portugal.

Object Managemeng Group (OMG) (2012). SysML Specification 1.3. http:// www.omgsysml.org/.

Object Managemeng Group (OMG) (2014). OMG Website. http://www.omg.org.

Object Management Group (2013a). Requirements Interchange Format Specification 1.1. http://www.omg.org/spec/ReqIF.

Object Management Group (2013b). Unified Modeling Language Specification. http://www.uml.org/.

Open Services for Lifecycle Collaboration (OSLC) (2014). OSLC Website. http://open-services.net/.

Polarion Software (2014). Polarion alm. https://www.polarion.com/products/alm/index.php.

Sparx Systems (2014). Enterprise Architect. http://www.sparxsystems.de/.

Walderhaug, S., Stav, E., Johansenand, U., and Olsen, G. K. (2008). Traceability in model-driven software development. *In: Tiako, P.F. (ed.) Designing Software-Intesnvie Systems: Methods and Principles*, pages 133–159.

Willert Software Tools (2014). ReqXChanger DataSheet. http://www.willert.de/assets/Datenblaetter/DatBl-ReqXChanger-V-2.0en-2014.pdf.

Winkler, S. and von Pilgrim, J. (2010). A survey of traceability in requirements engineering and model-driven development. *Software & Systems Modeling*, 9(4):529–565.