Comprehensive Model Integration for Dependency Identification with EMFTrace[*]

Stephan Bode, Steffen Lehnert, Matthias Riebisch
Department of Software Systems / Process Informatics
Ilmenau University of Technology
Ilmenau, Germany
{*stephan.bode, steffen.lehnert, matthias.riebisch*}*@tu-ilmenau.de*

***Abstract*—As model-based software development becomes increasingly important, the number of models to express various aspects of software at different levels of abstraction raises. Meanwhile evolutionary development and continuous changes demand for explicit dependencies between involved models to facilitate change impact analysis, software comprehension, or coverage and consistency checks. However, there are no comprehensive approaches supporting models, dependencies, changes, and related information throughout the entire software development process. The approach presented in this paper provides a unified and model-spanning concept with a repository for model integration, model versioning, and dependency identification among models utilizing traceability techniques, enhanced with analytic capabilities. The identification is based on a rule set to provide high values for precision and recall. The approach is implemented in a tool called EMFTrace, which is based on Eclipse technology and supports different CASE tools for modeling.**

***Keywords*-model dependencies; model integration; model repository; meta model; traceability**

I. Introduction

The growing complexity of software is often accompanied by difficulties in implementing changes and by architectural erosion. One possibility to tackle these problems is the use of different models and modeling languages throughout the entire development process to express requirements, design decisions, and dependencies between models, which is commonly referred to as model-based development. The wide use of several modeling languages and CASE tools has led to a variety of models with different scope and levels of abstraction. Integrating the various models and detecting and analyzing dependencies between them has become important.

It is therefore necessary to provide support for software evolution and continuous changes through models at different levels of abstraction and by explicitly expressing dependencies between them. Traceability links are well-suited to model such dependencies, since they enable further analyses, as change impact analysis, software comprehension, and consistency checks. Moreover, solid tool support is required to cope with the number of modeled entities and dependencies to supply practical applicability and usability.

Existing approaches provide solutions for the integration of models into centralized repositories, e.g., [6]. Besides, several different concepts have been proposed for dependency analyses on models through traceability rules, e.g., in [1]–[4], whereas [5] relies on information retrieval. However, these approaches only support a limited number of modeling languages without presenting a solution for spanning the whole software development process. They lack a unified concept for managing models, dependencies between models, types of dependency relations and auxiliary information in a centralized manner. This unified concept should be enforced to span the entire development process, ranging from early requirement definitions up to regression testing with the help of appropriate models. Heterogenous development environments and the wide use of several different CASE tools amplify the benefits gained from a unified and process-spanning approach.

The concept presented in this paper is aimed at creating an extensible platform for various analyses techniques required for model-driven engineering, reengineering, and impact analysis to support software evolution and continuous changes. It is based on a unified approach for comprehensive model integration and dependency analysis. In our approach inter and intra model dependencies are identified and recorded through traceability links, which are established by explicitly defined rules. These traceability rules are combined with information retrieval algorithms to achieve high precision and recall as well as flexibility. Special emphasis has also been put on the explicit modeling of dependency types, since they provide important semantic information for further analyses. Moreover, our concept provides a unified treatment of all related models and data through automated versioning and management of models, rules, model dependencies, and dependency types within a centralized repository. We utilize the extensible model repository EMFStore [6], which supports the integration of arbitrary models of standardized modeling languages such as UML, URN [7], BPMN and OWL. Several analysis components were integrated into EMFTrace to validate dependencies and to facilitate further analysis capabilities.

The remainder of this paper is organized as follows. We discuss current development and research regarding model repositories and model dependencies in Section II, whereas Section III introduces our concept and describes our approaches for model integration and dependency analysis. Section IV outlines our main conclusions and future work.

[*]The research presented in this paper was partly funded by the federal state Thuringia and the European Regional Development Fund ERDF through the Thüringer Aufbaubank under grant 2007 FE 9041.

II. State of the Art

A. Model Repositories

Automated model management and versioning can be achieved by model repositories that therefore assist model-based development of software. We considered several different repository projects as a base for EMFTrace, including the Eclipse Model Repository[1], CDO Model Repository[2], EMFStore[3] and AndroMDA[4]. Maturity, supported features, usability, and documentation were the main decisive factors we based our evaluation on. After a careful consideration of all projects, we decided to use the EMF[5]-based EMFStore as our underlying repository.

B. Detection of Model Dependencies

Since we focus on model-based development and dependencies between models, we do not consider approaches for dependency detection on the level of source code. Data gained through dependency analyses can either be stored and maintained in a dependency matrix, as a graph of traceability links, or as a set of cross-references [8]. Traceability links allow to attach auxiliary information to them, such as design decisions and alternatives, or link types. This information is of an equivalent value as the actual dependency itself, since it increases the semantical meaning of a dependency relationship. Therefore, it is most suitable to represent model dependencies with traceability links for further analyses.

Most traceability approaches focus either on predefined rules or on information retrieval. Information retrieval approaches as proposed in [5] or [9] operate on identifiers and are therefore easy to adopt to new models and data. They provide good recall, but their lack of precision and the complete absence of additional dependency information limit their usability for dependency detection among models and further analyses.

In contrast, rule-based approaches require more work to adopt to new models, but they result in more reliable links. Rule-based approaches as proposed in [1]–[4] are able to provide auxiliary information on dependencies, such as the type of the dependency specified by the rule. Well-defined dependency types are inevitable to identify semantically rich dependencies and to classify them properly.

III. EMFTrace

A. Concept and Features

EMFTrace implements our approach of comprehensive model integration and dependency identification as an extensible platform, which is our basis for further analyses and research. Our main focus is to provide a set of analyses and validation features, which is independent of a certain CASE tool and modeling language, to support engineers, software architects and other stakeholders throughout the entire software lifecycle. Based on models and dependencies between them, we want to establish solid tool support for the following activities:

- Architecture comprehension,
- Dependency analysis,
- Change impact analysis,
- Goal-oriented decision support for:
  - Forward engineering and
  - Reengineering; as well as
- Validation and consistency checks.

We utilize the EMFStore model repository for storing and versioning EMF-based models. Figure 1 provides an overview of the entire approach and illustrates the interaction of EMFTrace with external CASE tools.

To enable the integration and dependency analysis of different models we extended EMFStore by several metamodels as shown in the bottom of Figure 1. We integrated different modeling languages into the repository to span the entire software development process. Our approach supports requirements engineering through URN goal models, URN use case maps, and UML use cases. Furthermore, design models such as class diagrams are provided by UML, which has been fully integrated as well. Additional dependencies between UML and URN models are provided by the integration of OWL ontologies, which supply semantic networks of related terms. To ensure practical applicability, we support several different CASE tools which can be used to create and edit the corresponding models. More details on the integration of the models are given in Section III-B2.

As mentioned in Section II our approach utilizes rule-based traceability for dependency identification among models. To enable a unified treatment of all models and related data, we integrated additional metamodels into EMFStore to maintain and store traceability links and rules directly in the repository. Our traceability metamodel provides explicit support for modeling a hierarchy of dependency types to ensure the proper categorization of detected dependencies. Dependency types can be created and maintained by users in the repository and aggregated in special catalogs, and are therefore subject of model versioning as well. The metamodel for traceability rules offers a similar concept, which enables users to create rules directly in the repository and group them together in catalogs, to support fast exchange and easy maintenance. The design of our rules is based on similar approaches as proposed in [3] and [4]. However, we enhanced our rules with information retrieval techniques which supply additional query-operators to improve dependency detection. More details on the concepts for dependencies and rules follow in the next subsections.

1) Dependency Concept: We store dependency information as traceability links to benefit from their rich semantics. Our traceability metamodel supports binary and n-ary traceability links. Each traceability link can be enhanced with additional information such as the type of a dependency, design decisions, design alternatives, and

[1]http://modelrepository.sourceforge.net/project-summary.html
[2]http://wiki.eclipse.org/CDO
[3]http://www.eclipse.org/proposals/emf-store/
[4]http://www.andromda.org/index.php
[5]http://www.eclipse.org/modeling/emf/

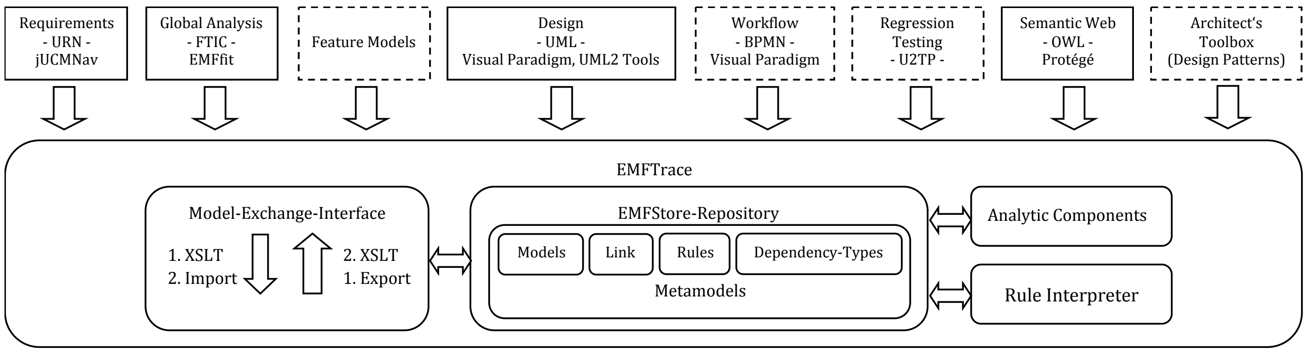

Figure 1. Conceptual overview of EMFTrace and its central role as a platform for various coupled research activities. The integration of dashed items is subject of current research and development.

assessments of the confidence of the link performed by users.

Moreover, we are able to combine transitively related traceability links, i.e., links that share common model elements as their source or target elements, into traces. These traces store chains of links to span design decisions and multi-level dependency relations. The example shown in Figure 2 illustrates the transition from a URN goal to its architectural realization through UML models. Furthermore, EMFTrace provides automated validation features for traceability links and traces, to ensure the integrity of stored dependency information. Corrupt or outdated links will be erased while broken traces are either split into smaller sub-traces or removed as well, if they contain less than two links.

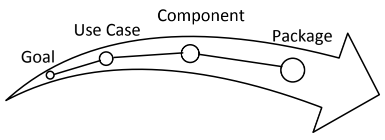

Figure 2. A trace containing a transitive chain of links

2) Rule Concept: Following the general design proposed in [3] and [4], our rules comprise three parts as shown in Figure 3, whereas each part is responsible for a certain task.

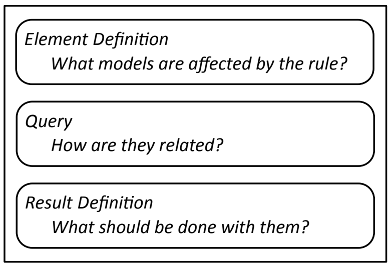
Element Definition
What models are affected by the rule?
Query
How are they related?
Result Definition
What should be done with them?

Figure 3. Main components of our rules

In contrast to existing approaches as [2], we do not operate on files, but on the conceptual level of models inside the EMFStore repository. Each rule can be equipped with several actions carried out once the conditions are met. This enables us to create a link for a relationship and its inverse relationship at once, without requiring additional rules. Rules can be created and edited by users in the repository using EMF-based editors or external in their XML representation, and can be imported afterwards. Rules can be grouped in catalogs to enable the use of specialized rule catalogs for different tasks such as dependency detection or consistency validation, to improve the useability of our tool. Currently there is one catalog available which contains 68 rules for traceability detection. Our rules operate directly on model attributes and the structure of models. They compare attributes to identify dependencies between models, for example the name attribute of an OWL class with the name attribute of an UML component. But since the comparison for exact matching is not always sufficient to detect all dependencies, information retrieval techniques, such as n-gram based string comparison, are used to compute the similarity of attributes.

B. Architecture and Realization

1) Architecture: Since EMFTrace is our evolving basis for many different research activities related to model-driven engineering, its architecure must support the addition of new concepts and features. Our entire tool is based on Eclipse technology and consists of several plug-ins that provide the core functionality, a user interface, and our adapted metamodels. EMFStore provides a server (the repository) and a client to operate with the repository, which is why we integrated our additional features into the client to provide a unified view and usability. We designed several additional components that integrate new features into the EMFStore client, such as an interpreter for traceability rules as shown in the bottom right corner of Figure 1. Our architecture is extensible and allows for easy integration of new components into EMFTrace to introduce new features, i.e., new analysis components. All plug-ins and components were implemented and tested with Java to ensure the platform independent applicability of our tool [10].

2) Model Integration: Using a modeling language with EMFTrace requires the integration of its metamodel into EMFStore. The metamodel must be converted into an

EMF-based Ecore model, which can either be modeled with EMF or generated from an XSD file. Once an Ecore model has been generated, it must be adapted to EMFStore's metamodel, which provides additional attributes to facilitate the management through EMFStore. We performed these steps for the metamodels of UML, URN, and OWL.

Since the adaptation to EMFStore's metamodel introduces new attributes to the metamodels, an adaptation of models (metamodel instances) is required as well, to add these attributes while importing models from CASE tools. We perform the integration and adaptation as illustrated by the arrows in the middle of Figure 1 by applying XSLT templates on models and encapsulated the entire import and export procedures in new components (*Model-Exchange-Interface*), which have been integrated into EMFTrace. Therefore, XML-based models exported by CASE tools and XSLT provide the core technology for our integration approach.

As CASE tools (see top of Figure 1) we currently support the Eclipse UML2Tools (UML), Visual Paradigm (UML, BPMN), the Eclipse-based jUCMnav (URN), and Protégé (OWL) with our XSLT templates. Additional modeling languages and their respective CASE tools such as BPEL can be added to EMFTrace at any time through adding an EMF-based metamodel to the underlying EMFStore repository and providing further XSLT templates.

3) Extensions: One addition which has already been integrated into EMFTrace is the tool EMFfit [12] (see Figure 1, top left), which supports the management of factor tables and issue cards for Global Analysis as proposed by Hofmeister et al. [11] and therefore contributes a new metamodel to EMFTrace. The tool itself is implemented and tested as a set of Eclipse plug-ins with Java and offers support for the transition from requirements to architectural design.

IV. Conclusion and Further Work

We presented an approach and a prototype tool EMFTrace for model integration and dependency identification between models of the entire software development process. We support different artifacts ranging from requirements models to design models which are managed by a unifying repository. Our tool provides comprehensive and automated dependency identification through rule-based traceability to ensure decent precision of results, enhanced with information retrieval techniques. EMFTrace facilitates the use of semantically important link types and is capable of storing, versioning, and maintaining traceability links. Standard modeling languages, such as UML, URN, and OWL, are integrated into the repository and several CASE tools are supported.

Further work will focus on several issues: additional modeling languages such as U2TP and feature models shall be integrated into EMFTrace along with the appropriate metamodels and XSLT templates, allowing for additional CASE tools to be used in conjunction with our tool. New models enable new analysis capabilities and require new rules or existing rules to be refined. Our rule concept should be enhanced to support consistency checks of models stored in the repository. The combination of dependency identification with consistency analysis provides the basic features to perform change impact analysis on models through new components and rules. To improve the usability of our tool, model synchronization with CASE tools and methods of change recognition for maintenance are considered as further enhancements. Drawing more benefits from discovered dependency relations requires sophisticated visualization of thereby created traceability links to enable the user to navigate on hierarchical chains of dependencies and to trace models efficiently. Finding appropriate means of dependency visualization is therefore one important research question related to EMFTrace.

References

[1] G. A. A. C. Filho, A. Zisman, and G. Spanoudakis, "Traceability approach for i* and UML models," in *Proceedings of 2nd International Workshop on Software Engineering for Large-Scale Multi-Agent Systems (SELMAS'03)*, 2003.

[2] W. Jirapanthong and A. Zisman, "Xtraque: traceability for product line systems," *Software and Systems Modeling*, vol. 8, no. 1, pp. 117–144, 2009.

[3] G. Spanoudakis, A. d'Avila Garces, and A. Zisman, "Revising rules to capture requirements traceability relations: A machine learning approach," in *Proc. Int. Conf. in Software Engineering and Knowledge Engineering (SEKE 2003)*. Knowledge Systems Institute, Skokie, 2003, pp. 570–577.

[4] G. Spanoudakis, A. Zisman, E. Perez-Minana, and P. Krause, "Rule-based generation of requirements traceability relations," *JSS*, vol. 72, no. 2, pp. 105–127, 2004.

[5] A. D. Lucia, F. Fasano, R. Oliveto, and G. Tortora, "Recovering traceability links in software artifact management systems using information retrieval methods," *ACM Trans. Softw. Eng. Methodol.*, vol. 16, no. 4, Article 13, Sept. 2007.

[6] M. Koegel and J. Helming, "EMFStore: a model repository for EMF models," in *Proc. Int. Conf. on Software Engineering (ICSE'10)*. ACM, 2010, pp. 307–308.

[7] ITU-T, "Recommendation ITU-T Z.151 User requirements notation (URN) – Language definition," ITU-T, Nov 2008.

[8] S. Winkler and J. von Pilgrim, "A survey of traceability in requirements engineering and model-driven development," *Softw. and Syst. Model.*, vol. 9, no. 4, pp. 529–565, 2010.

[9] A. Marcus and J. I. Maletic, "Recovering documentation-to-source-code traceability links using latent semantic indexing," in *Proc. Int. Conf. on Software Engineering (ICSE'03)*. IEEE, 2003, pp. 125–135.

[10] S. Lehnert, "Softwarearchitectural Design and Realization of a Repository for Comprehensive Model Traceability," Diploma thesis, Ilmenau University of Technology, Ilmenau, Germany, November 2010.

[11] C. Hofmeister, R. Nord, and D. Soni, *Applied Software Architecture*. Boston, MA, USA: Addison-Wesley, 2000.

[12] P. Wagner, "Tool Support for the Analysis during Software Architectural Design," Bachelor thesis, Ilmenau University of Technology, Ilmenau, Germany, December 2010.