Process for Applying Derived Property Based Traceability Framework in Software and Systems Development Life Cycle

Saulius Pavalkis and Lina Nemuraite

Department of Information Systems, Kaunas University of Technology,
Studentu 50-313a, Kaunas, Lithuania
saulius.pavalkis@nomagic.com, lina.nemuraite@ktu.lt

Abstract. For implementing the idea of applying derived properties for tracing project artifacts, the Derived Property Based Traceability Framework was created that consists of Model-Driven Domain Specific Language (DSL) engine for extending UML with derived property specifications, traceability schemas, and traceability analysis means. Traceability schemas may be generic, suitable for every purpose, but they often are characteristic to a development method, modeling language or a particular project. The paper presents a process for applying the Derived Property Based Traceability Framework consisting of three parts: process for adapting Derived Property Based Traceability solution for development method or Domain Specific Language; process for applying the solution in a development process, and process for automating the maintenance of traceability relations. Process is illustrated with examples from several case studies.

Keywords: traceability, derived properties, model-driven development, traceability framework.

1 Introduction

Traceability of software and systems models is an important aspect of Model Driven Development. Current state of traceability implementations in CASE tools often lacks flexibility, customizability and other qualities, analyzed by many authors and our previous works [1]. Usually, traceability solutions cause significant overhead and require routine efforts what often discourages from using traceability means at all.

We have proposed the traceability solution [1], based on derived properties, which is directed for solving frequent traceability problems. In particular, traceability solutions lack for automation; they pollute models with traceability information that can be redundant, burdening specification and analysis; additional relationships introduce dependencies and tight coupling among project stages that are incompatible with principles of good architectural design; traceability schemas are hardly customizable and maintainable.

The Derived Property Based Traceability Approach helps to avoid these problems as traceability relations are automatically calculated by a CASE tool when they are needed for analysis or validation of models. Derived attributes and relations of model elements are accessible for developers and analysts in specifications, dialog windows, visualization and analysis means in the same way as primary ones; so they do not require additional skills or specific attention.

The proposed traceability solution involves a traceability metamodel, profile, and the overall framework for implementing the solution [1], which is independent from a particular CASE tool. However, developers may wish to create specific traceability schemas for their chosen development methodologies and/or modeling languages as traceability schemas depend on types of modeling concepts and relationships, which are intended to trace.

Therefore, the goal of the paper is to present a complete process for ensuring traceability including adaption of the framework for different cases and automation of maintaining traceability relations. The overall process for using Derived Property Based Traceability approach consists of three parts: a process for adapting the solution for a particular methodology or language; process for applying the adapted solution in development projects; and a process for automating maintenance of traceability relations. We do not present here the traceability metamodel, profile, framework etc., as such information is available in [1] and [2]; instead, we illustrate the process with traceability schemas, validation rules etc., when needed.

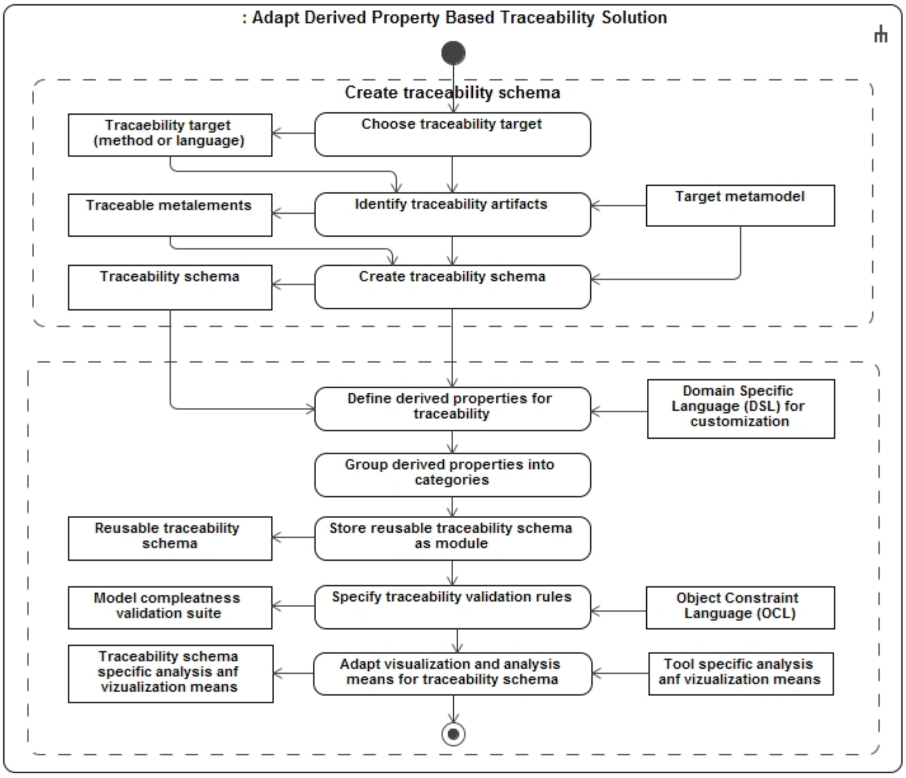
The rest of the paper is structured as follows. Sections 2 – 4 present the process for adapting, applying and automating derived property based traceability means in CASE tools. Section 5 provides overview of experimental approval. Section 6 analyses related work and gives a comparison of the approach with existing capabilities of similar tools. Section 7 presents conclusions and future works.

2 Process for Adapting Derived Property Based Traceability Solution

Process for adapting Derived Property Based Traceability solution is shown in Fig. 1. During creation of a traceability schema for a chosen modeling language, development process or a problem, one has to identify traceable artifacts and create derived properties for traceable links among these artifacts.

Choose Traceability Target. Any modeling language or development methodology can be selected as a traceability target. E.g. it could be the SysML [3] for specifying requirements, BPMN [4] for business analysis, and UML for software design. Standard or custom development processes (e.g. UP or SYSMOD [5]) can be used.

Identify Traceable Artifacts. In this step artifacts, whose evolution through project stages should be traced, are identified. Unless this is a mission critical system or different requirements are specified by standard regulations, usually only main artifacts, which influence stage or project completeness, are in focus. Too many artifacts will introduce overhead for managing traceability. Traceability rules are created for each relation between main artifacts, which relations are decided to be tracked. In order to achieve two–way traceability, traceability rules are created for deriving properties of both ends of traceability relations. Examples of such artifacts are BPMN Process, UML Use Case, SysML Requirement.



: Adapt Derived Property Based Traceability Solution
Create traceability schema
Tracaebility target (method or language)
Choose traceability target
Traceable metalements
Identify traceability artifacts
Target metamodel
Traceability schema
Create traceability schema
Define derived properties for traceability
Domain Specific Language (DSL) for customization
Group derived properties into categories
Reusable traceability schema
Store reusable traceability schema as module
Model compleatness validation suite
Specify traceability validation rules
Object Constraint Language (OCL)
Traceability schema specific analysis anf vizualization means
Adapt visualization and analysis means for traceability schema
Tool specific analysis anf vizualization means

Fig. 1. Process for adapting Derived Property Based Traceability solution

Create Traceability Schema. In order to create traceability schema, metaclasses of artifacts identified in the previous step are associated with tracing relations. Properties reflecting these associations will be owned by associations itself and will make no influence on standard modeling language or process metamodels. Example of traceability schema is presented in Fig. 2.

Define Derived Properties for Traceability. Simple expressions can be used to specify derived properties based on direct relationships, e.g. "Use Case → Satisfy → Requirement". The advanced Metachain expression should be used for transitive relations, e.g. "Business process → Abstraction → Use case → Abstraction→ Requirement → Satisfy → Component". OCL expressions and scripting languages should be used in more complex cases, e.g. for specifying recursive relations. An example of OCL expression for derived relation between component and use case (Fig. 2) is presented in Fig. 3.

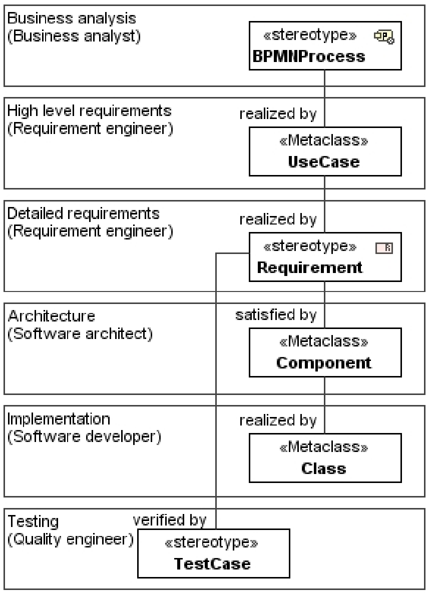


Fig. 2. Traceability schema for software development process

```
context BPMNProcess::RealizedInArchitecture:Component
  derive:self.supplierDependency→select(a|a.oclIsKindOf(Abstraction)).
    client→exists(b|b.oclIsKindOf(UseCase)).supplierDependency→
  select(c|c.oclIsKindOf(Abstraction)).client→exists(d|d.oclIsKindOf
    (SysML::Requirement)).supplierDependency→select(e|e.oclIsKindOf
        (SysML::Satisfy)).client→exists(f|f.oclIsKindOf(Component))
```

Fig. 3. OCL expression for derived property

Group Derived Properties into Categories, e.g. specification and realization groups (if traceability relation is established between artifacts of business process and its implementation, we treat traceability rules as realization ones; if we are going from implementation to business process, we consider them as specification rules).

Store Reusable Traceability Schema as a Module. Traceability schemas (sets of traceability relations) are dependent on traceability context – e.g. modeling language such as BPMN or software engineering process. It is desirable to keep traceability schemas in UML profiles and implement them as separate modules that could be loaded and reused in various projects. Derived properties defined in the loaded module are added to elements of considered models.

Specify Traceability Validation Rules. On the base of traceability schemas we can create validation rules and automate model analysis for checking model completeness (finding model elements not covered with their realizing artifacts, or identifying redundant artifacts); ensuring absence of cyclic traceability relations (i.e. such relations when e.g. one element is involved in both realizing and specifying traceability relations with another element).

OCL allows not only to specify traceability rules, but also to execute them. Having predefined traceability validation rules and using validation engine it is possible to check project for model completeness and cyclic traceability relations. Completeness validation rules are created for checking completeness of traceability (coverage of artifacts), e.g. each Use Case should be traced by at least one Requirement (Fig. 4).

```
Context: UseCase::realizedBy:Requirement
  (not self.ownedElement→exists (e|e.oclIsKindOf(UseCase)))
    implies self.realizedBy→size()>0
```

Fig. 4. OCL expression for artifact completeness validation rule

Adapt Visualization and Analysis Means for a Particular Traceability Schema. There are multiple types of UML relationships, properties and custom tags that can be used for traceability visualization. In order to help to quickly visualize traceability, custom (derived) properties are treated in the same way as regular element properties and can be represented on diagrams, validated with validation engine, and inserted into generated documents. Traceability property groups are visible in Element Specifications, Quick Properties, Go To, Reports, etc. Traceability information is available in Relation Maps for multi-level graph type traceability analysis; Dependency Matrix may be used for visualizing single level traceability and analyzing gaps. In order to be able to efficiently create and use traceability visualization means they can be predefined and distributed together with traceability schema.

3 Process for Applying Derived Property Based Traceability Solution

Process for applying Derived Property Based Traceability solution is shown in Fig. 5.

Apply Traceability Schema for Project. If traceability schema is held in a separate module (i.e. reusable project part) it can be loaded in a project and used starting from the beginning of the project or at any moment of already going project. If reusable traceability schema comes together with validation suites and visualization means, tree main immediate changes are observed on traceability module used in the project: 1) traceability properties appear in element specifications, context menu, and other places, and are immediately evaluated; 2) validation suites (if automatic) check model for completeness; incomplete and redundant artifacts are shown; 3) traceability visualization and analysis means (Dependency Matrix, Relation Map Dedicated reports, Generic tables are available and ready to be used).

Perform Coverage Analysis. The Coverage analysis gives coverage information at immediate higher (e.g., Specification) or lower (e.g., Realization) levels having the objective is to visualize and verify that artifacts of different stages, e.g., analysis, design, and implementation, are covered. It allows finding areas of not covered parts and to evaluate coverage metrics, to improve an understanding of the system and acceptance of the system accordingly.

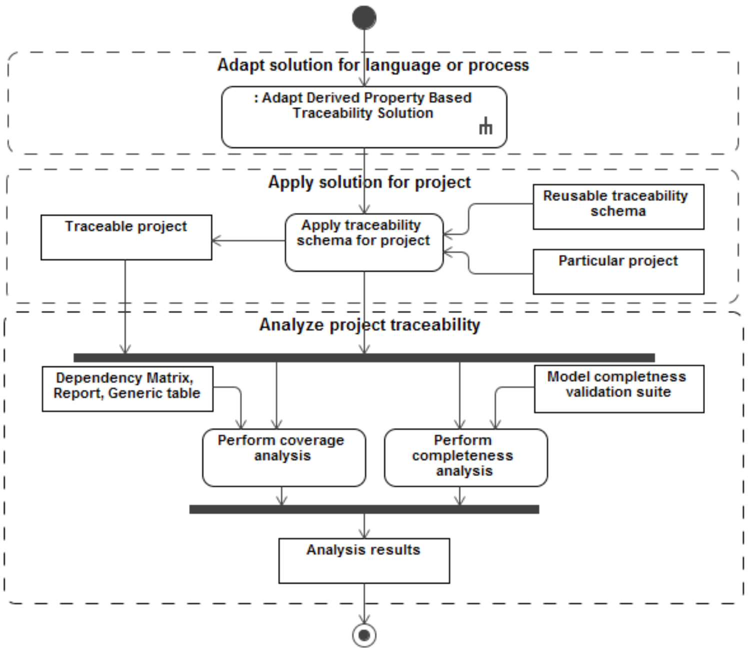

Adapt solution for language or process
: Adapt Derived Property Based Traceability Solution
Apply solution for project
Reusable traceability schema
Traceable project
Apply traceability schema for project
Particular project
Analyze project traceability
Dependency Matrix, Report, Generic table
Model completness validation suite
Perform coverage analysis
Perform completeness analysis
Analysis results

Fig. 5. Process for applying Derived Property Based Traceability solution

Calculate Traceability and Coverage Metrics. Examples of coverage metrics, which could be calculated for requirements of the overall system or level *n*:

1. The percent F_r of requirements in level *n* derived from requirements at level *n+1*:

$$F_r = \frac{R'_{n,n+1}}{R_n} 100\%$$

Here $R'_{n,n+1}$ is a number of requirements derived from requirements in level *n+1*; R_n – a number of all requirements in level *n*.

2. The percent O_r of requirements in level *n* excluding orphans derived from requirements at level *n+1*:

$$O_r = \frac{R'_{n,n+1}}{(R-O)_n} 100\%$$

Here $R_{n,n+1}$ is a number of requirements derived from requirements in level *n+1*; $(R–O)_n$ – a number of all requirements in level *n* excluding orphans.

3. The percent V_r of requirements in level *n* that are not verified (have no incoming *verify* relations):

$$V_r = \frac{V_n}{R_n} 100\%$$

Here V_n is a number of requirements in level *n* that are not verified.

4. The percent S_r of requirements in level n that are not satisfied by functions (used only at Sub-system level):

$$S_r = \frac{S_n}{R_n} 100\%$$

Here S_n is a number of requirements in level n that are not satisfied by functions (i.e. having no incoming *Satisfy* relationships from *PrincipleSet* or *Activity*).

5. The percent S_r of requirements in level n that are not satisfied by structural elements (i.e. having no incoming *Satisfy* relationships from *System*, *Subsystem*, *Product*, etc.):

$$SE_r = \frac{SE_n}{R_n} 100\%$$

Here SE_n is a number of requirements in level n that are not satisfied by structural elements.

6. The percent ST_r of requirements in level n that are not covered with Safety and Tests (i.e., have no outgoing trace relationships to requirements in level $n+1$) :

$$ST_r = \frac{ST_n}{R_n} 100\%$$

Here ST_n is a number of requirements in level n that are not are not covered with Safety and Tests.

Perform Completeness Analysis. It is possible to evaluate model against validation rules, which are checked automatically in all model or in a certain scope on demand. Results of validation rules evaluation show model elements, properties and diagrams, which does not satisfy validation constraints. One can see areas not yet covered with artifacts – incomplete ones, and redundant artifacts.

4 Automating Traceability Solution

Using derived property approach, traceability relations are automatically evaluated by derived property engine via calculating derived property values. However, without automation means for creating and updating derived properties, the approach would have a significant overhead, which would greatly discourage its usage in projects. Process for adapting and applying the Framework for Creating Custom Wizards (FCCW) [6] for automation of creating traceability relations and updating traceability information is shown in Fig. 6.

Choose Development Process for Automation. It is the first step in automating traceability. In the paper [6] two examples are presented about applying the proposed method for RUP-based workflow for use case modeling and capturing robustness analysis classes.

Create Process Workflow. Workflow, which will be automated, should be specified using Software Process Engineering Metamodel v2.0. In particularly, Process diagram needs to be used.

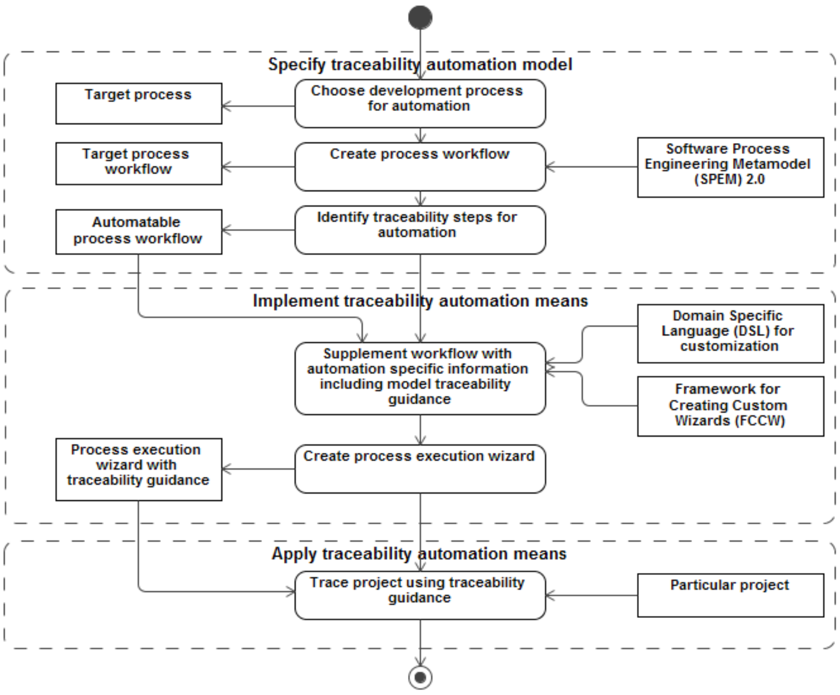

Specify traceability automation model
Target process
Choose development process for automation
Target process workflow
Create process workflow
Software Process Engineering Metamodel (SPEM) 2.0
Automatable process workflow
Identify traceability steps for automation
Implement traceability automation means
Domain Specific Language (DSL) for customization
Supplement workflow with automation specific information including model traceability guidance
Framework for Creating Custom Wizards (FCCW)
Process execution wizard with traceability guidance
Create process execution wizard
Apply traceability automation means
Trace project using traceability guidance
Particular project

Fig. 6. Process for applying FCCW for automating creation and updating of traceability relations

Identify Traceability Steps for Automation. Further, process steps, which will be automated, are identified. FCCW allows having 4 types of automation: creating an element which symbolizes the target model and defines its name; capturing elements and defining their properties, and listing elements existing in a model; joining elements with editable matrix like table to represent element relations; informing, navigating and invoking other features.

Supplement Workflow with Automation Specific Information. The specified process steps, which will be automated, are stereotyped with FCCW specific stereotypes identifying required automation type. Execution dedicated properties of each step are specified.

Create Process Execution Wizard. On the base of the specified workflow, FCCW stereotypes and their property information, executable wizard specification is created. To be generated wizard will guide through the workflow of traceability creation, analysis and update according to the chosen methodology, providing step-by-step dialog for tracing and creating elements. The wizard output is a model, from which further artifacts can be created: views, documentation, coverage analysis reports, etc.

Trace Project Using Traceability Guidance. The specified wizard can be included into a reusable traceability module together with traceability schema, validation rules and predefined view information. The wizard provides automation for gathering data according rules of modeling language and visualizing, creating and maintaining traceability.

5 Experimental Approval

Three experiments were conducted for evaluating the suitability of the Derived Property Based Traceability approach for implementing traceability solutions for software and system development processes, and modeling language BPMN. The experiments have shown that the approach is capable ensuring consistency of project artifacts, to analyze change impact, and to avoid typical traceability problems for software development [1] and for systems development [7] processes. It is capable to solve traceability problems of BPMN 2 models [8]: lack of traceability between BPMN processes and resource roles, BPMN processes and business concepts, participants and messages, thus allowing validating BPMN 2 models for correctness and completeness, and performing change impact analysis;

Discussion of Threats to Validity. The major threat of validity of the approach is an overhead raised by applying any traceability approach. This thread is eliminated in mission critical projects (e.g. health care, military, nuclear engineering, aerospace) in which traceability is of the great importance. The threat could be minimized in regular projects if only major artifacts are traced, and traceability automation means are used such us editable matrix for traceability relations, traceability validation suites, etc.

Another threat is the reliability of traceability validation results. Even approach is straightforward its results depend on how well it is followed. Also, if we validate coverage of major artifacts, we would not validate a quality of covering artifacts. To do so, validation constraints need to be extended to validate the content of covering artifacts.

6 Related Works

Early empirical studies showing importance of traceability for validating completeness of software or system projects have been published by Gotel and Finkelstein [9], Watkins and Neal [10], Ramesh and Edwards [11]. Aizenbud-Reshef et al. [12] emphasized the importance of automating traceability. We noticed three research directions for automatic creation and maintenance of traceability links: 1) Text mining and information retrieval techniques for recovering traceability links between software artifacts (e.g., [13]–[14]); 2) Establishing traceability links by monitoring users' modifications and analyzing change history; 3) Deriving traceability links from existing ones. The latter principle as less time consuming was used in our Derived Property based approach. We have supplemented it with two additional possibilities for reducing a manual input in creation and maintenance of traceability relations and obtaining a higher usability:

- Creating traceability information during model transformations. Automatic creation of traceability relations during transformation is analyzed in [13], [14]–[17]. As transformations are especially popular in Model Driven Engineering [18]–[25], we treat relations created during transformations as traceability ones.
- Analysis of existing relationships to obtain implied relations [26]. In our approach, a part of traceability information is based on transitive relations.

Comparison of existing traceability solutions in CASE tools with implementation of Derived Property Based Traceability approach in MagicDraw is presented in the Table 1:

Table 1. Comparison of existing traceability solutions in CASE tools

Criteria / CASE tool	**Rational Soft-ware Architect**	**Visual Para-digm**	**Enterprise Architect**	**Mode-lio**	**Reqtify**	**MagicDraw**
1. Traceability schema and rules are easy customizable and model driven	–	–	–	–	+/–	+
2. Capabilities of modeling tool are reusable for traceability analysis and visualization	+	+	+	+	–	+
3. Model is not polluted by traceability information	+	+/–	–	–	+	+
4. Model is loosely coupled	+	+/-	–	–	+	+
5. Creation and maintenance of traceability relations is automatic and flexible	+/–	+/-	–	+/-	+	+/–
6. Suggests traceability schemas	+	+/-	–	–	+	+
7. Coverage/completeness/change management analysis.	+/+/–	–/–/–	–/–/–	+/+/+	+/+/+	+/+/+

Analysis of existing traceability based solutions in CASE tools has shown that the presented solution provides advantages against other currently existing ones. The only equal solution with a similar number of steps to adapt to a custom development method is supported by the non-modeling tool – Geensoft Reqtify. Unfortunately, it requires programmatic integration with a modeling tool and adoption to a custom development method, what is not easy to achieve.

7 Conclusions and Future Works

The use of the proposed process by the real life examples for systems and software modeling projects and BPMN language has shown that the presented process provides the complete, development method independent methodology for adapting, using and automating the proposed traceability solution based on derived properties making it available for every model driven CASE tool.

Implementation of the approach in UML CASE tool MagicDraw has approved the expected quality criteria and was favorably met by MagicDraw users. It may be accomplished much faster and easier in comparison with traceability solutions of other CASE tools, which analysis revealed the advantages of the proposed process. The only equal solution with a similar number of steps to adapt to custom development method is supported by non-modeling tool – Geensoft Reqtify but it requires programmatic integration with a modeling tool and adaption to a custom development method, what is not easy to achieve.

Derived Property Based Traceability Approach already has been successfully adapted by companies including large aerospace and telecommunication corporations and academic institutions.

In our future work, we are planning to deepen our approach on the base of acquired practical experience: to automate transition from traceability metamodel to derived properties as this step could be fully automated; to help creating required traceability solutions by validating non-traced elements and automatically suggesting required relations to be created by using validation engine; to develop more powerful, comprehensive traceability schemas for modeling databases, business processes and enterprise architectures, which would be reusable across a large variety of software projects.

Acknowledgements. The work is supported by the project VP1-3.1-ŠMM-10-V-02-008 "Integration of Business Processes and Business Rules on the Basis of Business Semantics" (2013-2015), which is funded by the European Social Fund (ESF).

The authors would like to thank No Magic, Inc, especially the MagicDraw UML product team for the comprehensive support.

References

1. Pavalkis, S., Nemuraite, L., Butkiene, R.: Derived Properties: A User Friendly Approach to Model Traceability. Information Technology and Control 42(1), 48–60 (2013)
2. No Magic, Inc. UML Profiling and DSL (2011), `https://secure.nomagic.com/files/manuals/UML%20Profiling%20and%20DSL%20UserGuide.pdf`
3. OMG. OMG Systems Modeling Language (OMG SysML), Version 1.2. OMG, OMG Document Number: formal/2010-06-01 (2010)
4. OMG. Business Process Model and Notation (BPMN), Version 2.0. OMG, OMG Document Number: formal/2011-01-03 (2010)
5. SYSMOD, The Systems Modeling Process (2011), `http://sysmod.system-modeling.com/`
6. Silingas, D., Pavalkis, S., Morkevicius, A.: MD Wizard - a model-driven framework for wizard-based modeling guidance in UML tools. In: Proceedings of the International Multiconference on Computer Science and Information Technology, pp. 609–615. IEEE Computer Society Press, Los Alamitos (2009)
7. Pavalkis, S., Nemuraite, L.: Lightweight Model Driven Process to Ensure Model Traceability and a Case for SYSMOD. In: 2013 2nd International Conference on Advances in Computer Science and Engineering (CSE 2013), pp. 2019–2223. Atlantis Press (2013)
8. Pavalkis, S., Nemuraite, L., Milevičienė, E.: Towards Traceability Metamodel for Business Process Modeling Notation. In: Skersys, T., Butleris, R., Nemuraite, L., Suomi, R. (eds.) Building the e-World Ecosystem. IFIP AICT, vol. 353, pp. 177–188. Springer, Heidelberg (2011)
9. Gotel, O.C.Z., Finkelstein, A.C.W.: An analysis of the requirements traceability problem. In: Proceedings of the 1st IEEE International Requirements Engineering Conference (RE 1994), pp. 94–101. IEEE Computer Society, New York (1994)
10. Watkins, R., Neal, M.: Why and how of requirements tracing. IEEE Softw. 11(4), 104–106 (1994)

11. Ramesh, B., Edwards, M.: Issues in the development of a requirements traceability model. In: Proceedings of the IEEE International Symposium on Requirements Engineering, pp. 256–259. IEEE Computer Society, New York (1993)
12. Aizenbud-Reshef, N., Nolan, B.T., Rubin, J., Shaham-Gafni, Y.: Model traceability. IBM Systems Journal 45(3), 515–526 (2006)
13. Antoniol, G., Canfora, G., Casazza, G., De Lucia, A., Merlo, E.: Recovering traceability links between code and documentation. IEEE Transactions on Software Engineering 28(10), 970–983 (2002)
14. Hayes, J.H., Dekhtyar, A., Osborne, J.: Improving requirements tracing via information retrieval. In: Proceedings of the 11th IEEE International Requirements Engineering Conference, pp. 138–147 (2003)
15. Mens, T., Van Gorp, P.: A taxonomy of model transformation. In: Proceedings of the International Workshop on Graph and Model Transformation (GraMoT 2005), March 27. Electronic Notes in Theoretical Computer Science, vol. 152, pp. 125–142 (2005)
16. Porres, I.: Rule-based update transformations and their application to model refactorings. Software and Systems Modeling 4(2), 368–385 (2005)
17. Van Gorp, P., Janssens, D., Gardner, T.: Write once, deploy N: A performance oriented MDA case study. In: Proceedings of the IEEE International Conference on Enterprise Distributed Object Computing, pp. 123–134 (2004)
18. Schmidt, C.: Model-Driven Engineering. IEEE Computer 39(2), 25–31 (2006)
19. Briand, L.C., Labiche, Y., Yue, T.: Automated traceability analysis for UML model refinements. Information and Software Technology 51(2), 512–527 (2009)
20. Anastasakis, K., Bordbar, B., Georg, G., Ray, I.: On Challenges of Model Transformation from UML to Alloy. Journal on Software & System Modeling 9, 69–86 (2010)
21. Meijler, T.D., Nytun, J.P., Prinz, A., Wortmann, H.: Supporting fine-grained generative model-driven evolution. Software and Systems Modeling 9(3), 403–424 (2010)
22. Bryant, B.R., Gray, J., Mernik, M., Clarke, P.J., France, R.B., Karsai, G.: Challenges and Directions in Formalizing the Semantics of Modeling Languages. Computer Science and Information Systems 8(2), 225–253 (2011)
23. Recker, J.: Modeling with tools is easier, believe me – The effects of tool functionality on modeling grammar usage beliefs. Information Systems 37, 213–226 (2012)
24. Ablonskis, L., Nemuraitė, L.: Discovery of complex model implementation patterns in source code. Information Technology and Control 39(4), 291–300 (2010)
25. Lopata, A., Ambraziunas, M., Gudas, S., Butleris, R.: The main principles of knowledge-based information systems engineering. Elektronika ir Elektrotechnika 4(120), 99–102 (2012)
26. Sherba, S.A., Anderson, K.M., Faisal, M.: A Framework for Mapping Traceability Relationships. In: Proceedings of the 2nd International Workshop on Traceability in Emerging Forms of Software Engineering, Montreal, Canada (September 2003)