Lightweight Model Driven Process to Ensure Model Traceability and a Case for SYSMOD

Saulius Pavalkis[1], Lina Nemuraite[2]
Department of Information Systems. Kaunas University of Technology,
Kaunas, Lithuania
[1]saulius.pavalkis@nomagic.com, [2]lina.nemuraite@ktu.lt

Abstract—**When various software and systems development methodologies may be used in an organization, a problem becomes the process how to ensure project traceability independently from a chosen development method. Current state of traceability implementations in CASE tools lack flexibility, customizability and other qualities. The purpose of the current paper is to present a traceability process, independent from development methodology, and demonstrate how developers may apply this process to ensure a desirable traceability in their projects by using a lightweight approach, based on derived properties and general purpose traceability means, implemented in Model Engineering Environment of their CASE tools.**

Keywords-derived property based traceability; traceability; metamodel; completeness validation

I. Introduction

The continuously growing complexity and requirements for usability are inherent in modern "systems of systems" where software and other kinds of systems comprise the united whole. Examples of such complex systems are the European Extremely Large Telescope, modern car or locomotive, etc. On the other hand, the level of required usability is already raised by modern software such as Apple iOS. The holistic nature of system and software projects requires high skills and automation.

Large and complex projects require multiple means to manage complexity: acceptable time, appropriate team of professionals, good communication between team members, knowledge preservation in a case of team member change, development method to manage complexity, project traceability analysis, etc. Organizations usually adopt development methods to their needs. However, independently from a method, a project manager should easily track completeness of a whole project once the project is simultaneously changed by different roles. Also, all project members should see their position regarding completeness of their tasks in the context of the whole project.

Despite achievements in model driven engineering [1], current state of traceability implementations in CASE tools lacks flexibility, customizability and other qualities, analyzed by many authors [2]–[7], [8], [9], [10] and our previous works [11]. In particular traceability information pollutes models (traceability information can be redundant at specific system stage specification or analysis) with additional relationships that introduce dependencies and tight coupling among project stages; traceability schemas are hardly customizable and maintainable, so a care of traceability usually causes additional overhead.

We have proposed the traceability solution [11], based on derived properties, which is directed for solving these traceability problems and is implemented in UML CASE tool MagicDraw. The proposed traceability metamodel, profile, and overall framework are independent from a particular CASE tool. However, developers may wish to create the specific traceability schema for development methodology and/or modeling language as the schema depends on types of modeling concepts and relationships, which are intended to trace. Also, the quality of concrete implementation of traceability means depends on existing capabilities of CASE tools.

We have presented the derived property approach for BPMN traceability in [12] and for custom software development methodology in [11]. The purpose of the current paper is to present a traceability process, independent from development methodology, and demonstrate how software and systems developers may apply this process to ensure model traceability in their projects by using a lightweight approach, based on derived properties and general purpose traceability means, implemented in Model Engineering Environment of their CASE tools.

The rest of the paper is structured as follows. Section 2 presents the proposed traceability process. Section 3 shows an example of applying the process for tracing requirements using the SYSMOD method for modeling systems. Section 4 analyses related works and gives a comparison of the approach with existing capabilities of similar tools. Section 5 presents conclusions and future works.

II. Derived Property Based Traceability Process

When various development methodologies may be used in an organization, a problem becomes the process how to validate model completeness independently from a chosen development method. We will show that this is possible with a straightforward model based traceability process.

Fig. 1 presents an essential part of the derived property metamodel for traceability that supports the idea (the complete metamodel is presented in [11]).

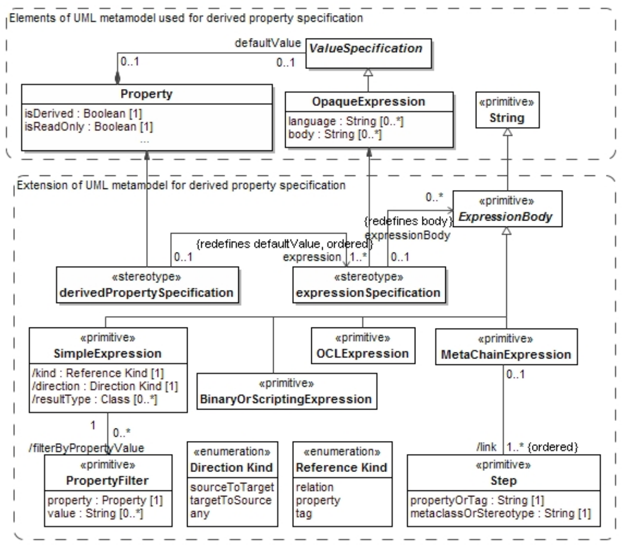

Figure 1. Part of the Derived Property Metamodel (excerpt from [11])

The Derived Property Metamodel extends UML metaclass "Property" with a stereotype <<derivedPropertySpecification>>. The latter has expression, stereotyped as <<expressionSpecification>>, for defining how this property is calculated. The stereotype <<expressionSpecification>> extends UML metaclass "OpaqueExpression" by redefining its properties "language" and "body". The property "body" of <<expressionSpecification>> is used to specify a primitive "ExpressionBody", which has several subtypes: <<SimpleExpression>>, multilevel <<Metachain Expression>>, <<OCLExpression>>, and <<BinaryOr ScriptingExpression>>. Such expressions may be supported by various UML CASE tools. Other expression types may be introduced as needed. The stereotypes are included into UML traceability profile, which comprises a part of a Model Engineering Environment.

In Model Engineering Environment, project tracing and completeness validation process consists of 5 steps presented in Fig. 2. These steps are:

- identification of major artifacts, whose evolution through project stages should be traced;
- creation of traceability schema – traceability relations among artifacts, which are dependent upon development methodology;
- specification of derived properties, which should be used for tracing relations among artifacts;
- creation of validation rules for checking coverage of traceable artifacts;
- checking validation rules and analyzing results (project completeness can be checked in any point of time by any role).

It is worth to note that such process may be defined for custom development methodologies and reused in many projects, or adopted for a specific project if needed.

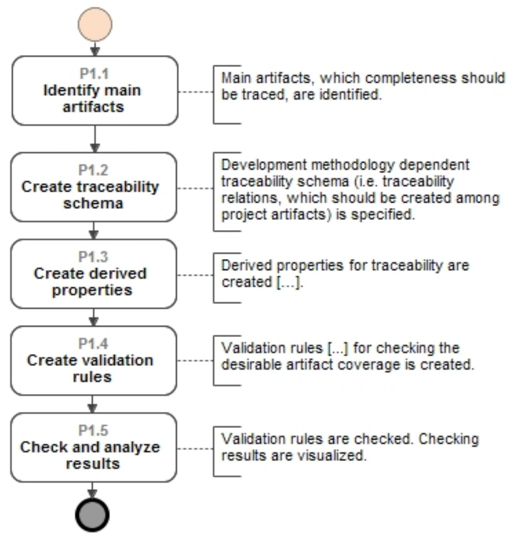
P1.1
Identify main artifacts
Main artifacts, which completeness should be traced, are identified.
P1.2
Create traceability schema
Development methodology dependent traceability schema (i.e. traceability relations, which should be created among project artifacts) is specified.
P1.3
Create derived properties
Derived properties for traceability are created [...].
P1.4
Create validation rules
Validation rules [...] for checking the desirable artifact coverage is created.
P1.5
Check and analyze results
Validation rules are checked. Checking results are visualized.

Figure 2. Project completeness validation process

III. TRACEABILITY ENSURING PROCESS EXAMPLE FOR SYSMOD

We will demonstrate step-by-step how project completeness validation process is adopted for SYSMOD [13] – a pragmatic approach for modeling requirements as well as a functional and physical architecture of a system. It provides a toolbox of tasks with input and output work products, guidelines and best practices using the OMG Systems Modeling Language (SysML) [14]. SYSMOD is used for creating an example of Car Access System model (available online http://example.system-modeling.com) using the CASE tool MagicDraw from NoMagic Inc.

A. *Identification of Traceable Artifacts*

We will concentrate on main SYSMOD artifacts (which completeness represents project or stage completeness) and relations among them created by roles participating in the project (Table I). It took us one hour to identify main artifacts (about 28.6% from the total time to ensure traceability).

TABLE1 ARTIFACTS IDENTIFIED FOR TRACING PROJECTS, WHICH USE SYSMOD

Discipline	Model	Artifact	Primary performer
Requirements analysis	System Requirements:: Objectives	Requirement	System analyst
Requirements analysis	System Requirements:: Essential	Requirement	System analyst
Requirements analysis	Domain knowledge model	<<Domain block>> Block	System analyst
Requirements analysis	Use Cases	<<System use cases>> Use Case	System analyst
System Architecture	System Breakdown	Block	System architect

B. Traceability schema

In order to create traceability schema for SYSMOD we will take metaclasses of artifacts identified in the first step and associate them with relations, which should exist between them as tracing relations. Properties reflecting these associations created for the intended traceability schema will be owned by associations itself and will make no influence on standard UML and SYSMOD metamodels. The traceability schema for SYSMOD is presented in Fig. 3 where associations between selected artifacts identify the desirable traces, role names represent their semantics.

We decided to skip optional artifacts from the traceability schema and analyze coverage and completeness of a model on the base of mandatory artifacts Objective, Essential Requirements and System Breakdown Block. The desirable completeness of the model is defined by rules "Each Objective should be traced by at least one Requirement" and "Each leaf Requirement should be satisfied by at least one Block". The first rule is presented in the Traceability schema by the multiplicity "1..*" of <<trace>> relation between Objective and Requirement. The second rule cannot be directly expressed by graphical notation. Both rules are specified in OCL in Subsection 3.D. It took us one hour to identify traceability schema (about 28.6% from the total time to ensure traceability).

Note: too many artifacts will introduce overhead with traceability management. Balance shall be maintained. It took one hour to decide, which artifacts are desirable for tracking their coverage by other artifacts in the further project stages. Fragments of artifacts of a Car Access System model are presented in Fig. 4.

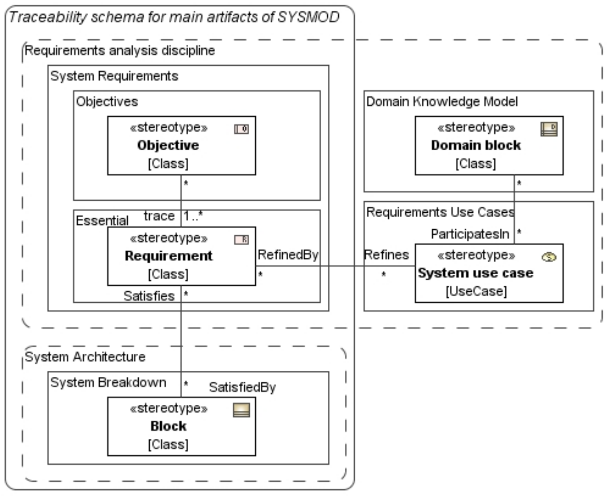
Traceability schema for main artifacts of SYSMOD
Requirements analysis discipline
System Requirements
Objectives
«stereotype» Objective [Class]
Essential
trace 1..*
*
«stereotype» Requirement [Class]
RefinedBy
Satisfies *
Domain Knowledge Model
«stereotype» Domain block [Class]
Requirements Use Cases
ParticipatesIn *
Refines *
«stereotype» System use case [UseCase]
System Architecture
System Breakdown
* SatisfiedBy
«stereotype» Block [Class]

Figure 3. Traceability schema for main artifacts of SYSMOD

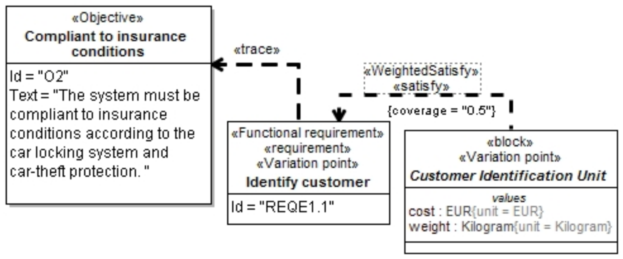
«Objective» Compliant to insurance conditions
Id = "O2"
Text = "The system must be compliant to insurance conditions according to the car locking system and car-theft protection. "
«trace»
«WeightedSatisfy» «satisfy»
{coverage = "0.5"}
«Functional requirement» «requirement» «Variation point» Identify customer
Id = "REQE1.1"
«block» «Variation point» Customer Identification Unit
values
cost : EUR{unit = EUR}
weight : Kilogram{unit = Kilogram}

Figure 4. Example of traceable artifacts of Car Access System model

C. Derived Properties for Traceability

According to the traceability schema we will identify traceability rules and create derived properties for tracing mandatory artifacts (Table II). We used simple expressions for specification of traceability rules (derived relations <<trace>> and <<satisfy>> among artifacts of the <<weightedSatisfy>> stereotype adds a property to calculate a rate of coverage). It took us 10 minutes to create derived properties for traceability (about 4.8% from the total time to ensure traceability).

TABLE2 DERIVED PROPERTIES AND TRACEABILITY RULES FOR CAR ACCESS SYSTEM MODEL

#	Rule name	Source element	Expression	Target element
1	Trace	Objective	Trace	Requirement
3	Satisfied By	Requirement	Satisfy	Block

D. Validation rules for checking derived properties

Once derived properties are specified they appear in specifications of corresponding elements and other places in the same way as regular UML properties. Now we can validate model completeness by performing coverage analysis for discovering whether all objectives are covered by requirements and requirements satisfied by design or not. The following rules specified for validating the desirable completeness of chosen artifacts are presented in Table III.

TABLE3 VALIDATION RULES IN OCL TO CHECK COMPLETENESS OF TRACEABILITY

No	Rule header	OCL Expression
1	Context Objective	(not self.ownedElement→ exists(e|e.oclIsKindOf (SysML::Requirement))) implies derive:self.trace→size()>0
2	Context Requirement	((not self.oclIsKindOf (SYSMOD::Objective)) and (not self.ownedElement→ exists(e|e.oclIsKindOf (SysML::Requirement)))) implies derive:self.satisfiedBy→size()>0

OCL constraints check the presence of values of chosen traceability rules. The first rule checks if all objectives of the project (specified as SysML objectives) are covered by requirements. The second rule checks if all leaf requirements of the project are satisfied by architectural elements of the system. If some traceability rule has a value it means that traceability relation and coverage exist; otherwise, the artifact is uncovered.

E. Model validation

Now we can evaluate model against validation rules, which are checked in a certain scope (i.e. package marked with stereotype <<Validation suite>>). After some iteration of improvements (that took about one hour of our time – about 28.6% from the total time to ensure traceability) the validation rules and scope of validation gave expected results and became suitable for checking the desirable completeness of project artifacts. The final analysis of results after correction took 20 minutes – 9.4% from the total time to ensure traceability.

F. Process implementation

The Derived Property Based Traceability means are implemented in UML CASE tool MagicDraw reusing its Domain Specific Modeling Environment, Customization Engine, and other tool's capabilities [15]–[17], [7].

IV. CONCLUSIONS AND FUTURE WORKS

The paper has presented the development method independent process for adopting the proposed traceability solution based on derived properties, and how it was applied for the custom method for modeling systems – SYSMOD. The use of the proposed process was demonstrated for the real life example – Car Access System model.

The presented example has shown that our model driven traceability approach, based on derived properties, is straightforward and easy path for checking the desirable completeness of project models. It has taken less than 3.5 hours to set means for tracing given project artifacts and validating their coverage. As the derived property based traceability approach is implemented in UML CASE tool MagicDraw, the process is actually used by MagicDraw users. Of course, preparation for checking coverage of project artifacts in practice depends on project and might require slightly more efforts than in the presented case; nevertheless, it may be accomplished much faster and easier in comparison with other solutions. The only equal solution with a similar number of steps to adopt to custom development method is supported by non-modeling tool – Geensoft Reqtify but it requires programmatic integration with a modeling tool and adoption to a custom development method, what is not easy to achieve.

Once adopted, validation means allows validating models for completeness or tracking progress in any point of time as often as needed. Validation is accomplished in a single step without requiring for an additional time.

ACKNOWLEDGEMENT

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

REFERENCES

[1] C. Schmidt, "Model-Driven Engineering," in IEEE Computer, vol. 39(2), pp. 25–31, 2006.

[2] O. C. Z. Gotel, A. C. W. Finkelstein, "An analysis of the requirements traceability problem," in 1st IEEE International equirements Engineering Conference (RE'94) Proceedings, pp. 94–101, IEEE Computer Society, New York, 1994.

[3] R. Watkins, M. Neal, "Why and how of requirements tracing," IEEE Softw, vol. 11(4), pp. 104–106, 1994.

[4] B. Ramesh, M. Edwards, "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.

[5] N. Aizenbud-Reshef, B. T. Nolan, J. Rubin, Y. Shaham-Gafni, "Model traceability," IBM Systems Journal vol. 45 (3), pp. 515–526, 2006.

[6] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, E. Merlo, "Recovering traceability links between code and documentation" IEEE Transactions on Software Engineering vol. 28 (10), pp. 970–983, 2002.

[7] J. H. Hayes, A. Dekhtyar, J. Osborne, "Improving requirements tracing via information retrieval," in Requirements Engineering Conference, 2003, Proceedings, 11th IEEE International, pp. 138–147, 2003.

[8] L. C. Briand, Y. Labiche, T. Yue, "Automated traceability analysis for UML model refinements," in Information and Software Technology, vol. 51(2), pp. 512–527, 2009.

[9] T. D. Meijler, J. P. Nytun, A. Prinz, H. Wortmann, "Supporting fine-grained generative model-driven evolution," Software and Systems Modeling, vol. 9(3), pp. 403−424, 2010.

[10] S.A. Sherba, K.M. Anderson, M. Faisal, "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.

[11] S. Pavalkis, L. Nemuraite, R. Butkiene, "Derived Properties: A User Friendly Approach to Model Traceability," Information Technology and Control, Kaunas, vol. 42(1), pp. 48–60, 2013.

[12] S. Pavalkis, L. Nemuraite, E. Mileviciene, "Towards Traceability Metamodel for Business Process Modeling Notation," in IFIP Advances in Information and Communication Technology, vol. 353, 1868-4238. pp. 177–188. Heidelberg, Dordrecht, London, New York: Springer, 2011.

[13] "SYSMOD – The Systems Modeling Process," 2011, http://sysmod.system-modeling.com/

[14] OMG, "OMG Systems Modeling Language (OMG SysML)," Version 1.2, OMG, OMG Document Number: formal/2010-06-01, 2010.

[15] D. Šilingas, R. Butleris, "Towards Implementing a Framework for Modeling Software Requirements in MagicDraw UML," Information Technology and Control vol. 38(2), pp. 153–164, 2009.

[16] D. Šilingas, R. Vitiutinas, A. Armonas, L. Nemuraite, "Domain-specific modeling environment based on UML profiles," in Information Technologies' 2009: proceedings of the 15th International Conference on Information and Software Technologies, IT 2009, Kaunas, Lithuania, April 23-24, 2009, pp. 167-177. Kaunas University of Technology, Technologija, Kaunas, 2009.

[17] No Magic, Inc., "UML Profiling and DSL," 2011, https://secure.nomagic.com/files/ manuals/UML%20Profiling%20and%20DSL%20UserGuide.pdf