

A Framework for Requirements Traceability in UML-based Projects*

Patricio Letelier

Department of Information Systems and Computing
Technical University of Valencia (Spain)
letelier@dsic.upv.es

Abstract

Requirements traceability allows us to assure the continuous concordance between the stakeholders requirements and the artifacts produced along the software development process. Although the important role of requirements traceability is widely recognized, the application level and consensus about associated practices are quite variable from one software development team to another. UML appears as an opportunity to establish a common framework for requirements traceability. In this work we present a reference metamodel for requirements traceability, that is based on UML and integrates as much textual specifications as UML model elements, obtaining a homogeneous representation for all the software development artifacts and traceability links among them. Thanks to the UML extension mechanisms, we have obtained in a natural way that our approach be adaptable according to the project needs. We have included an example illustrating how to use our framework in a small project, taking Rational Unified Process (RUP) as a software development process.

1 Introduction

Software requirements are susceptible of changing, not only after releasing the product but also along the iterative software development process. Requirements management is the process in charge of supervising these software requirements changes, it should

be integrated as a subprocess in the software development process, being the core of the software process [1]. The success of this subprocess depends on how well defined the relationships among requirements and other kinds of specifications generated by the software process are. Requirements traceability is defined as the ability to describe and follow the life of a requirement in both directions, towards its origin or towards its implementation, passing through all the related specifications.

Requirements management and especially requirements traceability can be expensive activities. The detail level in these activities and the collected information must be configured according to the particular project needs, in order to obtain a positive cost-benefit ratio.

Nowadays the effectivity in traceability practices differs considerably among development teams. Some problems that can explain this situation are [9, 10]: there are not detailed guidelines regarding the kinds of information that must be gathered for traceability, the context in which such an information must be used, and the lack of consensus about the semantic for the links between specifications.

In reference to the way of expressing them, requirements have been traditionally specified using textual forms of specification above all, mainly using natural language. Consequently, tools supporting requirements management have been focused on the manipulation of text pieces. These textual-expressed requirements are linked forming a traceability graph

*This work is funded by the *DOLMEN-SIGLO* project of the "Comisión Interministerial de Ciencia y Tecnología" (CICYT), TIC2000-1673-C06-01.

which is used to manage the requirements and their traceability. In this approach, the specifications generated in other activities of the development process can also be added to the traceability graph, representing them as text (normally using the name of the specification, for instance: the name of the class, attribute or operation). Test specifications are also mainly textual, thus they can be handled in a similar way. Even though several CASE tool vendors claim that their products offer a good integration among modules for requirements management, modeling and test, the implemented solution is usually based on import/export mechanisms among such modules. This strategy is not the best one as far as its promulgation as a part of the software process is concerned. Another suggested alternative is to specify explicitly in additional diagrams the traceability links between model elements, but apart from not covering all kinds of specifications, due to the complexity of the traceability graph, this is not viable, even for small systems.

On the other hand, regarding the software modeling, UML [11] has quickly become the most popular notation for object-oriented modeling. Thanks to the definition of its metamodel and the included extension mechanism, UML offers an excellent opportunity to establish a common framework for representing specification of requirements, development and test.

The aim of this work is to present a framework for configuring requirements traceability by integrating textual specifications and UML model elements. Our approach can be applied to any software process based on UML and it can be customized according to the specific traceability needs of the project.

This paper is organized in seven sections. Following this introduction, the next section describes a metamodel for requirements traceability. The third section explains how textual specifications and traceability links can be defined in the UML context, obtaining a common framework for expressing all the traceability information. In section four we present the tasks for configuring the traceability in a project. The fifth section illustrates the application of our approach using a small project based on RUP as an example. The sixth section describes some related

works and specific tools for requirements management, from the perspective of frameworks for requirements traceability. Eventually, the seventh section presents our conclusions.

2 A metamodel for requirements traceability

Before presenting our metamodel for requirements traceability we will summarize the information needs for requirements management. Next we indicate the kinds of information associated to requirements traceability and their possible uses (adapted from [2]):

1. Traceability links between different types of specifications allow validating that the system functionality covers the stakeholders expectations, that there is not superfluous functionality implemented, and performing impact analysis when requirements change.
2. Contribution structures [3], that is, the links between stakeholders and specifications allow improving the communication and cooperation among stakeholders, and guaranteeing that the contribution of every individual is considered and registered.
3. Rationale associated to specifications, including alternatives, decisions, assumptions, etc. contribute to improve the understanding and acceptance of the system from the stakeholders, and to improve change management avoiding studying again considerations already excluded. This is possible thanks to making accessible the solutions, their fundamentals, and the excluded alternatives.

In Figure 1, by means of a class diagram, we present a metamodel for requirements traceability. Classes represent entity types and associations represent types of traceability links. We use role names attached to some association extremes to make more legible the types of traceability links.

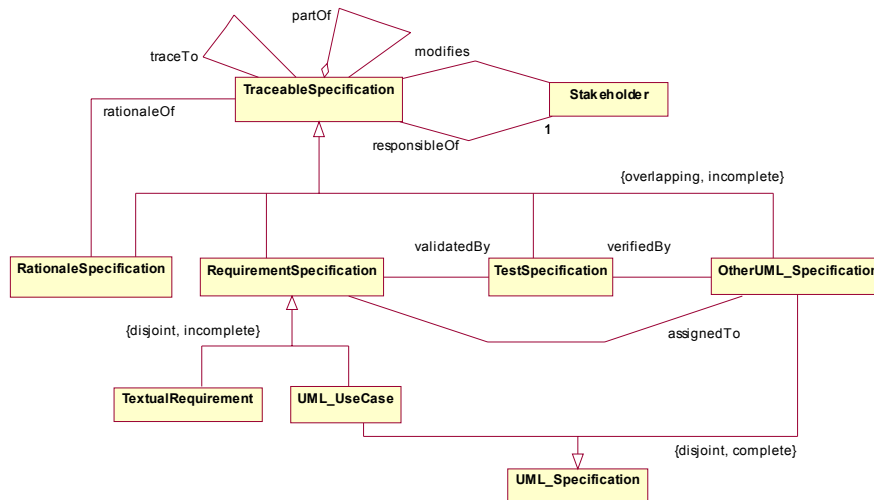


Figure 1: Metamodel for requirements traceability

Globally speaking, we are concerned with two types of entities: *TraceableSpecification* and *Stakeholders*. *Stakeholders* are responsible of creating and modifying specifications. A *TraceableSpecification* is a software specification with a certain granularity level, that is, it can be a document, a model, a diagram, a section in a document, a text specifying a non-functional requirement, a use case, a class, an attribute, etc. The granularity for a *TraceableSpecification* is defined by means of the aggregation with the role name *partOf*.

The type of entity *TraceableSpecification* is a generalization of *RationaleSpecification*, *RequirementSpecification*, *TestSpecification*, and *OtherUML_Specification*. A *TraceableSpecification* can belong to more than one of such subtypes, for instance, when a document includes several types of specifications (using *partOf*). A *RequirementSpecification* is a requirement or group of requirements. Requirements, according to how they are expressed, can be classified as *TextualRequirements* (requirements expressed using a text piece) or *UML_UseCase* (the corresponding UML model element for representing a functional requirement). A *RationaleSpecification* establishes, for instance: fundamentals, alternatives or assumptions associated to a *TraceableSpecification*. Eventually, a *TestSpecification* defines a test for val-

idating a requirement or verifying a UML model element (for instance: verifying a source code file that is the implementation of a class or component). Generalizations whose parent classes are *TraceableSpecification* and *RequirementSpecification* are defined as “incomplete” allowing the definition of other types of specifications that can be interesting for traceability. For instance, some specifications that are neither textual nor standard UML are: videos, images, voice, etc. These specifications usually correspond to fundamentals, and are useful during the revision and assessment of analysis and design models [4]. However, they could also be a particular media for expressing a *RequirementSpecification*, for instance, registering it in a video.

The most generic type of traceability link is represented by the association with the role name *traceTo* which allows establishing traceability links between any *TraceableSpecification*. The rest of types of traceability links (*modifies*, *responsibleOf*, *rationaleOf*, *validatedBy*, *verifiedBy* and *assignedTo*) are more specific. The type of link named *modifies* allows establishing a relationship between *Stakeholders* and *TraceableSpecifications* that they modify. In a similar way, *responsibleOf* determines the *Stakeholder* who is responsible of the definition and maintenance of a *TraceableSpecification*. The type of link named *vali-*

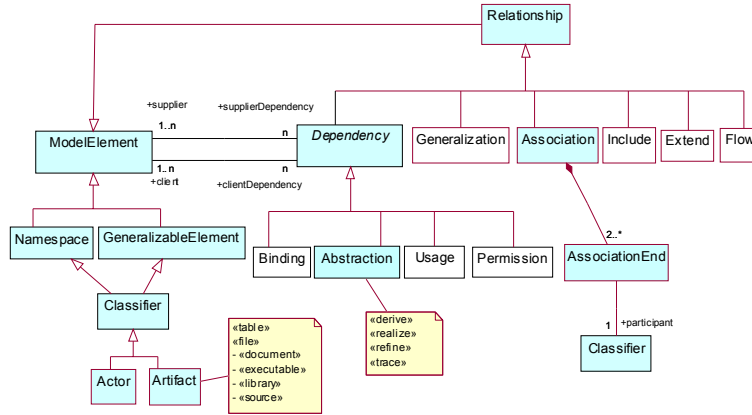


Figure 2: UML context for our traceability metamodel

datedBy relates *RequirementsSpecifications* with the corresponding *TestSpecifications* that validate them. The type of link *verifiedBy* determines the *TestSpecifications* that verify a UML specification. Finally, the type of link *assignedTo* determines the UML model elements that realize certain requirements, for instance, the classes that realize a use case.

The metamodel showed in Figure 1 covers the four perspectives of traceability information included in the works by Ramesh and Jarke [10]: requirements, rationale, allocation of requirements to model and implementation elements, and eventually, test. Moreover, our metamodel incorporates pre-traceability and post-traceability aspects [6, 13]. Pre-traceability allows going from the origins of the requirements until their explicit specification in the Software Requirements Specification (SRS) document or vice versa. Post-traceability allows going from the SRS to the subsequent software and test specifications or vice versa. In both kinds of traceability our metamodel provides the types of links *responsibleOf* and *modifies* to determine the *Stakeholders* involved. For pre-traceability the type of link *traceTo* is available between requirements expressed in different abstraction levels and *rationaleOf* for rationale associated to such requirements specifications. The post-traceability is supported by the types of links *traceTo*, *validatedBy*,

verifiedBy, *assignedTo* and *rationaleOf*.

3 UML context for the traceability metamodel

To make simple and practical the application of our metamodel it is convenient to integrate all types of entities and links in a common context. Considering that: (a) UML specifications are more precisely defined and they are more accepted than other specifications, (b) UML provides extension mechanism (*stereotypes*, *tagged values* and *constraints*) to incorporate new types of specifications, and (c) UML specifications have support in most CASE tools, it is evident that it would be appropriate to integrate all types of specifications of our metamodel in the context of UML. Thus, for each type of entity and type of link a correspondence with a UML model element will be established. To do this, UML metaclasses will be chosen as base classes to establish new stereotypes. When the type of entity or type of link matches up semantically with a UML metaclass, this metaclass will be used directly without defining a new stereotype. The result of this analysis is a UML profile for our traceability metamodel. Next we give details about how such an integration is performed.

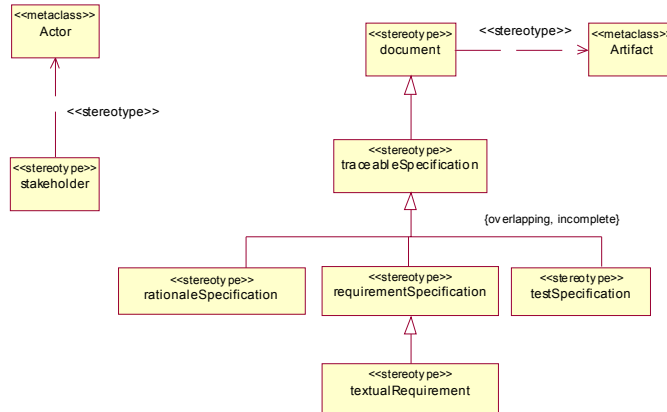


Figure 3: Stereotypes for *stakeholder* and core textual specifications

Traceability entities in the UML context. For the entity *Stakeholder* the choice is simple; the model element *Actor* is the metaclass used as a base class to define the corresponding stereotype. Other types of entities should have the possibility of allowing associations in order to establish aggregation relationships between them. According to this, the selected metaclass should be among the UML elements that are child classes of *Classifier*. For entities corresponding to non-UML standard specifications, the *Classifier* named *Artifact* (added in UML version 1.4) has been chosen. *Artifact* has some predefined stereotypes, among them «document», which is the one we will use to represent documents and document sections. For types of entities that match up directly with UML model elements (*UML_UseCase* and *OtherUML_Specification*) we will use the UML model element itself. On the other hand, to group and organize UML artifacts and *Stakeholders* we will use the UML model element to this end, that is, *Package* and optionally we will add the predefined stereotypes «model» or «subsystem», depending on whether we are defining a model of a system/subsystem or dividing the system in subsystems, respectively.

Traceability links in the UML context. Types of links will be represented as UML model elements of *Abstraction* type, except the relationship *partOf*, which is represented by aggregation or composition between specifications using the metaclass *Associa-*

tion as the base class. Although different types of links are modeled by different associations in our traceability metamodel, they are not independent, in fact, the type of link *traceTo* is a generalization of all other types of links. The type of link *traceTo* will be coincident with the stereotype «trace», predefined in UML. In UML a «trace» dependency indicates a historical or process relationship between two elements that represent the same concept without specifying derivation rules between them [11]. Excepting the type of link *partOf*, other types of links will be child stereotypes of the predefined stereotype «trace».

In Figure 2 we show the UML context for the concepts we are dealing with. According to the previous explanations, Figure 3 and Figure 4 show the UML representation for types of entities and types of links included in our traceability metamodel. This representation constitutes an essential UML profile for requirements traceability.

4 Configuring traceability

In requirements traceability we can identify two activities: (a) configuration of traceability according to the project needs, and (b) specify and exploit traceability information during the software development and maintenance. We will focus on the configuration activity applying our UML profile for traceability.

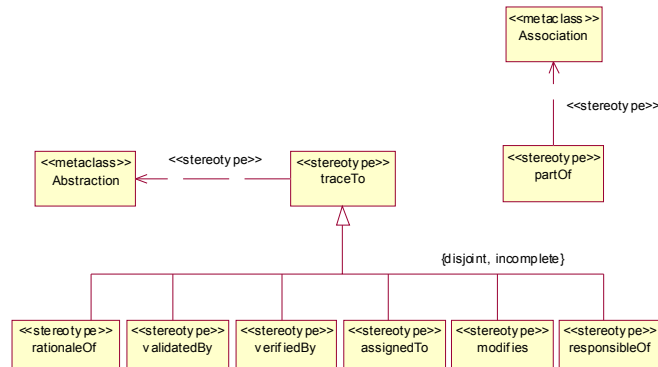


Figure 4: Stereotypes for the traceability links

The profile will act as a framework for establishing the types of artifacts¹ interesting for traceability and the types of links between them. Consequently, traceability links established during the software development or maintenance are verified against the traceability configuration (for instance, for a particular project, certain types of links are valid only between certain types of artifacts).

Configuring traceability in a project includes the following tasks:

1. Select types of artifacts which are interesting from the traceability perspective. They constitute a subset of the entire types of artifacts used by the project. Each selected artifact will have a corresponding stereotype as a child class of one stereotype defined in the traceability profile (except for those types of artifacts that are standard UML model element). When doing this, we will be extending the profile with new stereotypes for entities (this includes the possibility of defining new stereotypes as specializations of stakeholder).
2. Define aggregation relationships between artifacts. This task may not be necessary for all types of artifacts if such relationships are pre-defined and included in the description of types of artifacts.

3. Establish types of traceability links that are interesting to the project. The types of links are established between pairs of types of artifacts selected in task 1. In this case it may also be necessary to extend the traceability profile including new types of links as stereotypes specializations.
4. Define criteria to implicitly derive traceability links and what types of links (established in task 3) will use these criteria.

5 Configuring traceability in a RUP project

The presented metamodel and the corresponding profile are independent from the software development process, the only assumption is that the process is based on UML. However, to illustrate the application of our approach we have chosen RUP as a process, mainly because it offers enough details and variety regarding the available artifacts. RUP is a Rational Software product based on the Unified Software Development Process [5]. RUP provides templates Word and HTML for artifacts that are documents, and it uses UML for software modeling. Next we will illustrate the tasks for configuring the requirements traceability in a small RUP project.

¹From now on we will use the term “artifact” in a wider sense, beyond the definition provided in UML, and in the same way as most software processes do (for instance RUP). Thus, we will consider as artifacts all documents, files and other physical elements generated or used during the software development process, in addition, we will call artifacts any UML model element.

Task 1 Select the types of artifacts that will be traceable specifications. In this example we will use the RUP artifacts showed in the next table:

RUP Type of Artifact	Parent Class Stereotype
Vision	«traceableSpecification»
Software Feature	«textualRequirement»
Supplementary Spec.	«traceableSpecification»
Non-Functional Req.	«textualRequirement»
Assumption	«rationaleSpecification»
Use Case Specification	«traceableSpecification»
Use Case Precondition	«traceableSpecification»
Use Case Model	«model»
Use Case	
Analysis & Design Model	«model»
Class	
Implementation Model	«model»
Component	
Data Model	«model»
Test Case	«testSpecification»

When, in the table above, there is not a parent class stereotype this is because it uses directly the corresponding UML model element. As we mentioned before, the stereotype «model» is the UML package model element with such a stereotype.

Task 2 Define aggregation relationships between types of artifacts. For our selected artifacts the aggregation relationships are:

- Vision \diamond — Software Feature
- Vision \diamond — Assumption
- Software Feature \diamond — Software Feature
- Supplementary Specification \diamond — Non-Functional Requirement
- Use Case Specification \diamond — Use Case Precondition
- Use Case Model \diamond — Use Case
- Analysis & Design Model \diamond — Class
- Implementation Model \diamond — Component
- Data Model \diamond — Table

Task 3 Establish the types of traceability links that are interesting for the project. In our example they are:

Stakeholder $\xrightarrow{\text{«responsibleOf»}}$ RUP Artifact

Stakeholder $\xrightarrow{\text{«modifies»}}$ RUP Artifact
 Software Feature $\xrightarrow{\text{«traceTo»}}$ Use Case
 Assumption $\xrightarrow{\text{«supports»}}$ Software Feature
 Use Case $\xrightarrow{\text{«traceTo»}}$ Use Case Specification
 Use Case $\xrightarrow{\text{«validatedBy»}}$ Test Case
 Use Case Precondition $\xrightarrow{\text{«traceTo»}}$ Class
 Class $\xrightarrow{\text{«traceTo»}}$ Component
 Class $\xrightarrow{\text{«traceTo»}}$ Table
 Class $\xrightarrow{\text{«verifiedBy»}}$ Test Case
 Component $\xrightarrow{\text{«verifiedBy»}}$ Test Case

We have used *RUP Artifact* to refer to any RUP artifact selected in task 1. The stereotype «supports» has been introduced as a new stereotype specialization of the stereotype «rationaleOf».

Task 4 To establish implicit traceability we will use transitivity, aggregation relationships and exact name matching between types of artifacts. In this last case the criterion of name equality will only be applied to the following types of links:

Use Case $\xrightarrow{\text{«traceTo»}}$ Use Case Specification
 Use Case $\xrightarrow{\text{«validatedBy»}}$ Test Case
 Class $\xrightarrow{\text{«traceTo»}}$ Table
 Class $\xrightarrow{\text{«verifiedBy»}}$ Test Case
 Component $\xrightarrow{\text{«verifiedBy»}}$ Test Case

Eventually, according to the established traceability configuration, we could have at a certain instant the traceability graph showed in Figure 5. *FEATURE* is the type of artifact *Software Feature*, *UC* is the type *Use Case*, *UCS* is a *Use Case Specification* type, and *PRECOND* corresponds to the type *Use Case Precondition*. The link *Sales management : FEATURE* $\xrightarrow{\text{traceTo}}$ *Make an order : UC* is derived by the aggregation relationship between the software features *Sales management* and *Books sales*. The link *Make an order : UC* $\xrightarrow{\text{traceTo}}$ *Make an order : UCS* is derived by the name matching criteria. Other implicit transitivity or aggregation links are ignored because they are not interesting types of links (defined in task 3).

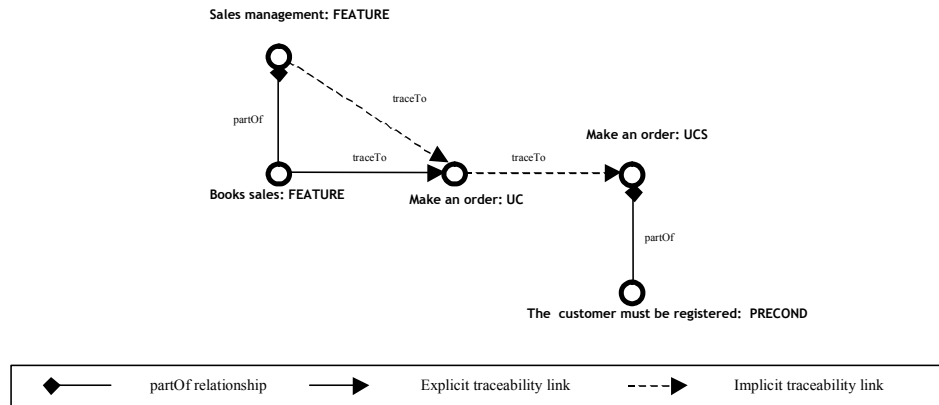


Figure 5: An example traceability graph

6 Related works

Ramesh and Jarke in [10] offer a wide vision about the information needed in requirements traceability. Their study is based on the analysis of industrial software development projects. They identify two segments of traceability users and suggest two corresponding traceability metamodels (one is a simplification of the other). The most complete metamodel has 31 types of entities (metaclasses in the metamodel) and about 50 types of links. In addition to the complexity associated to the diversity of types of entities and links, a precise definition for those elements is not provided, which makes the application of the metamodel a difficult task. Curiously, all analysis and design specifications are only represented by one type of entity named “*system_subsystem_component*”, that is, there is no more granularity or connection with any specific modeling notation. In this work the only suggested mechanism to configure the metamodel according to the project needs is to cut or to add parts of the metamodel.

Toranzo and Castro in [15] present a traceability metamodel defined by multiple views, each of them associated to a certain kind of user for the traceability information (*Project Manager*, *Requirement Engineer* or *Software Engineer*). In this work there is no configuration mechanism to adapt the traceability to

the project needs. Furthermore, the granularity level of the artifacts is too thick, working with documents and diagrams.

Spence and Probasco in [14] present several strategies for traceability when a use case driven process is used (as the case of RUP). Each strategy is described with a simple traceability metamodel, establishing the types of artifacts and links. All the strategies suggested only consider links between artifacts near to requirements (user needs, software features, use cases, etc.). The connection with artifacts for modeling and test is left implicit according to what a use case driven process establishes (use case analysis or design realization, functional test for each use case, etc.). Furthermore, the only one type of link they use is our equivalent *traceTo*. Similarly, Leite et al. in [7, 8] provide a framework for elicitation and organization of requirements expressed in natural language. They establish traceability links between requirements but they do not include traceability to other subsequent artifacts.

On the other hand, requirements management tools² offer a satisfactory treatment for textual specifications but they have inconveniences when integrating those specifications with others not expressed textually. This integration is based on import mechanisms connecting with a CASE tool. Usually, in this approach the names of the modeling elements are handled in the context of the requirements manage-

²To obtain a more detailed analysis we recommend to visit the INCOSE (Internacional Council On System Engineering) web site. www.incose.org/tools/tooltax.html

ment tool. A very cited tool is TOOR (*Traceability of Object-Oriented Requirements*), presented by Pinheiro and Goguen in [12], it is based on FOOPS, a formal object-oriented language. Curiously, there are not any works about TOOR after the mentioned paper. However, the formal approach and the functionality described remain interesting. In Rational RequisitePro (www.rational.com) textual specifications can be linked with UML model elements in the Rational Rose repository. But this is possible only for use cases, other model elements are not directly accessible from Rational RequisitePro. The only type of link in RequisitePro is our equivalent *traceTo*. Another well known requirements management tool is Telelogic DOORS (www.telelogic.com). This tool can be connected with most popular CASE tools, using similar import mechanisms of Rational RequisitePro, but providing more functionality and facilities to change from the DOOR context to the CASE tool context. Anyway, the user must work with two separated environments, and depending on whether he/she wants to do requirement management or software modeling, he/she must switch the environment. All the mentioned tools have inconveniences as far as the configuration of traceability to the project needs is concerned. They are not oriented to a specific software process and although some of them allow defining types of requirements, they do not offer a framework for configuring requirement traceability. Eventually, all the definitions and interpretation about the traceability information is left to the user of the tool.

7 Conclusions

Requirements traceability is the key to achieve a successful requirements management process. However, there is no consensus about the more suitable strategies to perform an effective requirements traceability. Thus in practice, requirements traceability presents different levels of satisfaction and acceptance in software development projects. Consequently, tool support for this activity is not appropriate. Historically, research on requirements management has been carried out in parallel to research on approaches to software modeling and development, they constitute two different communities: one community focused on improving the treatment of requirements specifications without going further as far as software modeling notations or methodology are concerned, the other one

focused on improving analysis and design techniques but not paying much attention to requirements specifications. Nowadays, UML, the most popular object-oriented notation, represents an excellent opportunity to take advantage of the results from both communities. UML can be used to define a common framework to support requirements and modeling specifications.

In this work we have presented a traceability metamodel integrating textual specifications (for requirements, rationale and tests) with standard UML specifications, using the UML context itself. Thus, from the point of view of requirements traceability, our metamodel offers a core framework for types of entities and types of traceability links that can be adapted to a particular project using the extension mechanisms provided by UML. The traceability metamodel has been translated to a UML profile to allow an easier application in a CASE tool supporting UML.

Additionally, we have sketched a configuration process for requirements traceability based on our UML profile for requirements traceability. Our approach including the metamodel, the corresponding UML profile and the configuration process only have the assumption of using a UML-based process, but it is independent of any particular process. However, to illustrate our approach we have presented an example using RUP as a development process. Currently we are working on the design and construction of a module that implements our approach, and according to the RUP process. This module will be integrated in Rational Rose.

An important aspect when configuring the traceability for a project is to establish traceability attributes (and their possible values) for each selected type of artifact. Deliberately we have not dealt with this topic in this paper because we consider that this is not especially difficult. It can be assumed that the requirements management tool will offer a set of predefined attributes for each type of artifact and the user will be able to select them or define new ones. For instance, in RUP some attributes for software features are: state (proposed, approved or incorporated), benefit (critical, important or useful), estimated effort, risk and stability (these attributes usually have values such as high, medium or low).

References

- [1] J.-P. Corriveau. "Traceability Process for Large OO Projects". IEEE Computer, pp. 63-68, September 1996.
- [2] R. Dömges and K. Pohl. "Adapting Traceability Environments to Project-Specific Needs". Communications of ACM, Vol. 41, No 21, December 1998.
- [3] O. Gotel and A. Finkelstein. "Extended Requirements Traceability: Results of an Industrial Case Study". In Proceedings of 3rd International Symposium on Requirements Engineering (RE97), IEEE Computer Society Press, pp. 169-178, 1997.
- [4] P. Haumer, M. Jarke, K. Pohl and K. Weidenhaupt. "Improving reviews of conceptual models by extended traceability to captured system usage". Interacting with Computers, Elsevier Science, 13 (1) pp. 77-95, 2000. <ftp://sunsite.informatik.rwth-aachen.de/pub/CREWS/CREWS-99-16.ps.gz>
- [5] I. Jacobson, G. Booch and J. Rumbaugh. *The Unified Software Development Process*. Addison-Wesley, 1999.
- [6] M. Jarke. "Requirements Tracing". Communications of the ACM, Vol. 41, No. 12, pp. 32-36, December 1998.
- [7] J.C. Leite and A.Oliveira. "A Client Oriented Requirements Baseline". In Proceedings of the Second IEEE International Symposium on Requirements Engineering (RE'95), pp. 108-115, IEEE Computer Society Press, 1995
- [8] J.C. Leite, G. Rossi, F. Balaguer, V. Maiorana, G. Kaplan, G. Hadad, A. Oliveros. "Enhancing a Requirements Baseline with Scenarios". Requirements Engineering Vol. 2, No. 4, pp. 184-198, 1997.
- [9] B. Ramesh. "Factors influencing requirements traceability practice". Communication of the ACM, Vol. 41, No. 12, pp. 37-44, December 1998.
- [10] B. Ramesh and M. Jarke. "Toward Reference Models for Requirements Traceability". IEEE Transactions on Software Engineering, Vol. 27, No. 1, pp.58-93, January 2001.
- [11] OMG Unified Modeling Language Specification. UML 1.4 with Action Semantics, Final Adopted Specification, January 2002. www.omg.org
- [12] F. Pinheiro and J. Goguen. "An Object-Oriented Tool for Tracing Requirements". IEEE Software, pp. 52-64, March 1996.
- [13] K. Pohl. "PRO-ART: Enabling Requirements Pre-Traceability". In Proceedings of the 2th International Conference on Requirements Engineering (ICRE'96), pp. 76-84, 1996.
- [14] I. Spence and L. Probasco. "Traceability Studies for Managing Requirements with Use Cases". Rational Software White Paper, 1998. www.rational.com/products/whitepapers/022701.jsp
- [15] M. Toranzo and J. Castro. "A Comprehensive Traceability Model to Support the Design of Interactive Systems". ECOOP Workshops 1999, pp. 283-284, Lecture Notes in Computer Science 1743, Springer-Verlag, 1999.