A Traceability Link Model for the Unified Process

Patrick Maeder, Ilka Philippow and Matthias Riebisch
Technical University of Ilmenau, Germany
patrick.maeder|ilka.philippow|matthias.riebisch@tu-ilmenau.de

Abstract

Traceability links are widely accepted as efficient means to support an evolutionary software development. However, their usage in analysis and design is effort consuming and error prone due to lacking or missing methods and tools for their creation, update and verification.

In this paper we analyse and classify Unified Process artefacts to establish a traceability link model for this process. This model defines all required links between the artefacts. Furthermore, it provides a basis for the (semi)-automatic establishment and the verification of links in Unified Process development projects. We also define a first set of rules as step towards an efficient management of the links. In the ongoing project the rule set is extended to establish a whole framework of methods and rules.

1. Introduction

Business critical software systems are highly complex and have to fulfil rapidly changing needs. These changes bear high risks, such as misunderstood dependencies, missing comprehension, incomplete implementation and lacking coverage. Evolutionary development processes have been developed to support these frequent changes to the system. One of these concepts is traceability. Most development process standards support traceability and mention it as one of its features. Today, traceability is mostly limited to relating requirements and poorly used in practice. But traceability links are necessary for the whole development process from requirements to the implementation of the system. Traceability links have to be kept in a correct and complete state and they have to be defined at a fine-grained level to be useful. This requires a very high number of links to be managed and maintained even for small systems. The maintenance and management has currently to be carried out manually and requires an extreme high effort. The precondition for an effective tool support is a detailed integration of traceability links into development methods. The definitions of activities, relations and artefacts of most design methods are too imprecise and vague to define rules for traceability links based on them and to give support for the concept of traceability.

In order to achieve results of high practical value, a widely used design methodology is applied as base of our work. We have chosen the Unified Process UP for the definition of a process-specific model of traceability links in this paper. For the Unified Process, there are rather detailed descriptions of the design methodology. Although, Letelier showed in [5] the application of his metamodel for the UP, his definitions are not detailed enough to derive rules for traceability links. There is no detailed description of how and between which artefacts traceability links should be established, although the UP description introduces traceability as one of its features. Additionally, a syntactic and semantic definition of traceability links is necessary.

As contribution of this paper we classify and analyse the UP artefacts concerning to traceability aspects. Based on that, all required links between the artefacts of the UP activities of requirements engineering and design are defined by providing a traceability link model. Additionally, a syntactic and semantic definition of traceability links is established customized to the UP's methods. The analysis of the UP and the customisation of the traceability concept are performed during practical development projects. As results of these works, rules for the verification of traceability links have been established.

2. Related Work

An overview of research topics, results and open issues in the field of traceability was given in a former publication [6]. In this paper according to the specific topic, three studies concerning traceability frameworks have to be investigated in particular.

Letelier [5] offers a metamodel for requirements traceability in UML-based projects. He gives an example of the usage in a UP project. The author is focusing on a general traceability model and gives advise on how to customize it using UML mechanisms. By keeping the model generally usable, it is not possible to define rules and activities for the creation, verification and the update of links, which could be performed (semi)automatically by a tool.

Spence and Probasco [8] discuss several alternatives for traceability between requirements. The paper is focused on the UP. The authors do not give answer the question how the transition to analysis and design and the following development steps should be traced.

Based on the analysis of industrial software development projects Ramesh and Jarke [7] define two metamodels for traceability. The authors differentiate low-end and high-end users of traceability. Correspondingly they provide a simplified and a full version of their metamodel. Furthermore, they establish a predefined standard set of link types. The authors focus especially on project management and on organizational needs of traceability. They do not solve the problem how traceability can be established in analysis and design.

3. Traceability

Traceability is the ability to follow and recover the development steps of a system based on the connection between inputs or stimuli of every development step with its products. These products, later called artefacts, are the inputs of next development steps. This leads to a graph of dependencies, which shows the realization of the systems requirements within the developed system.

Types of Traceability Links. There is a broad variety of types and concepts related to traceability, e.g. the concept of dependencies in UML and its extension in SysML [9]. For categorization purposes we introduce the notion of a traceability link type. A traceability link type classifies the relationship between two connected elements and/or the development activity for the generation of the destination element from the source element. Using types of traceability links aims to minimize the necessary number of different rules for establishing and checking of links. It is not necessary to find rules for every possible combination of source and destination elements, type-based rules can be applied instead. Based on our research we establish the following four types of traceability links as the basic ones:

- Refinement («refine») – in accordance with the level of detail of the connected objects (e.g. between an analysis and a design object).
- Realization («realize») – the dependent object represents a part of the solution to the problem described with the independent object (e.g. between a use case and an analysis class).
- Verification («verify») – of behaviour and properties of the developed solution or its parts (e.g. between a use case and a test case) and
- Definition («define») – of objects (e.g. between a glossary item and its usage in one of the models).

Traceability Link Representation. In the UML traces are defined as a special kind of dependency. Therefore, the same graphical representation is used: a unidirectional arrow, enhanced with the stereotype «trace». For a simple dependency the arrow is directed from the dependent (destination) to the independent (source) element e.g. an analysis object is connected toward a use case. The graphical direction of the traceability link does not exclude its usage in both directions, forwards and backwards.

4. The Unified Process UP

Software development processes consist of activities and artefacts leading from requirements to the systems implementation. The handling of traceability links can be the more tool supported the more the acts of a developer correspond to the activities of a method. It is possible to apply traceability rules to these activities. The better and the more fine-grained the process description is the easier is the definition of rules for creating and updating traceability links. Therefore, the approach for traceability proposed in this paper is presently focused on the Unified Process, because it is concrete, widely used and well described.

In the UP several ancestor methods like Object-Oriented Software Engineering OOSE [4] have been combined based on best practices and experiences. The UP is available as commercial and as open-source version. The UP process model, the activities of the method and the composition of the artefacts are described detailed enough for the aimed level of support. The UP can be customized and concretised to particular projects and companies needs. The UP is an incremental and iterative process: it is based on use case and architecture centric development of software. The incremental, iterative approach can be seen as a two-dimensional scheme as described in [1].

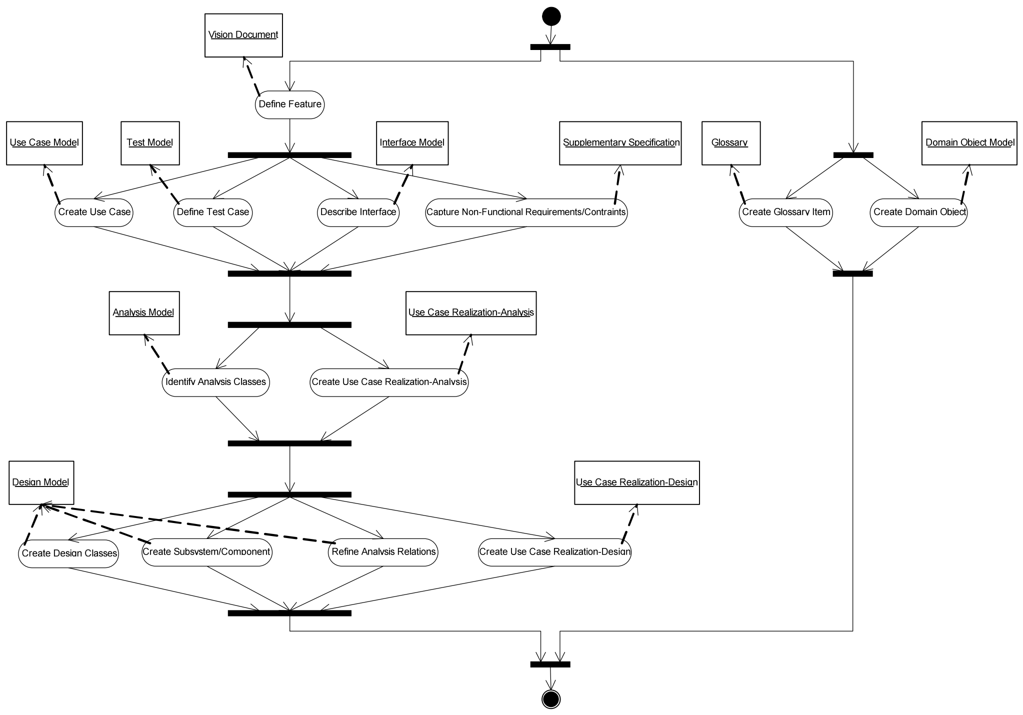
Vision Document
Define Feature
Use Case Model
Test Model
Interface Model
Supplementary Specification
Glossary
Domain Object Model
Create Use Case
Define Test Case
Describe Interface
Capture Non-Functional Requirements/Contraints
Create Glossary Item
Create Domain Object
Analysis Model
Use Case Realization-Analysis
Identify Analysis Classes
Create Use Case Realization-Analysis
Design Model
Use Case Realization-Design
Create Design Classes
Create Subsystem/Component
Refine Analysis Relations
Create Use Case Realization-Design

Figure 1. Development Activities of the UP Workflows: Requirements and Analysis/Design

4.1. Development Activities and Relations Between Model Elements

In this section a model of useful traceability links for the UP is proposed. At first each of the UP development activities is briefly explained and necessary traceability links between the involved artefacts are established. To give an overview, the major traceability related development activities of the UP are shown in Fig. 1 by an activity diagram. It has to be pointed out that a sequential representation of activities is used for better visualisation. However, in practice the activities are carried out incrementally in several iterations.

4.1.1 Development Activities during the Requirements Workflow.

Elaboration of the Vision Document. Based on a natural-language text document of stakeholder requirements (needs), the system features have to be defined. The needs and the realizing features are connected by explicit traceability links of the type «realize».

Creating the Glossary and the Domain Object Model. Parallel with the vision document the glossary elaboration has to be started by defining and entering all domain-relevant terms. Each new term identified during an activity must be defined, before it can be used. The developer has to ensure that there is not already another term defined for the same issue. If the new term has relations to other terms it has to be modelled in the DOM as well. Additionally, every term has to be categorized by one of the following types: actor, object or process. These categories refer to the type of term used within the before introduced templates and for the naming of model elements. By knowing the type of a term, it is possible to verify its correct usage within a text template or within an identifier of a model object.

Development of the Use Case Model. As first step the border of the system and the interacting actors must be specified. The actors have to be defined in the glossary as well. The next step is to find use cases for the before defined features. Between use cases and features m:n relations can exists, that means that several use cases can refine one feature or that several features are refined by one use case. Features and use cases are connected by an explicit traceability link of type «refine». The association between an actor and a triggered use case can lead to an implicit traceability link. The use case specification should be enhanced with test case specifications for the verification of its realization. Use cases and test cases have to be connected by an explicit traceability link of type «verify». The relation is of m:n multiplicity.

Development of the Interface Description. Textual documents, GUI-prototypes or models can be used for interface descriptions. The description of an interface contains associations between actors and use cases, in which an interface is used, represented by an explicit traceability link of type «refine».

4.1.2 Development Activities of OO-Analysis.

Identification of Analysis Classes. In the analysis phase classes and packages are used for modelling the structure of the system. In the UP analysis classes are distinguished as interface, entity or control class. There are different approaches for finding analysis classes. The examination of nouns and verbs in use case descriptions is a widely accepted technique. Nouns are candidates for classes or attributes and verbs are candidates for responsibilities or methods. Another way to find classes is the CRC-card method. The particular choice for a method is determined by the project. Every use case is connected by explicit traceability links to the analysis classes, which realize its flow. Each class can be connected to several or only one use case and vice versa. That means a class can realize more than one use case.

Performing of Use Case Realizations-Analysis. In this step the cooperation between the different analysis classes has to be described by UML interaction diagrams. For each use case at least one diagram is modelled, representing communication and messages between instances.

The interaction diagrams have to be connected with the related use case, using an explicit traceability link of type «realize». It is also possible to connect them implicit by using consistent diagram names. By drawing messages between classifiers in interaction diagrams an implicit connection between the corresponding classes is established. This connection can be used to verify associations in the class model between these classes.

4.1.3 Development Activities during Design.

Creation of Design Classes (Design Model). The design model is a refinement of the analysis model. As a first step all elements of the analysis model have to be copied. The copied elements are considered as initial design model. It is possible to connect analysis and design elements automatically while copying them by explicit traceability links of type «refine».

During the design phase almost all elements of the initial design model are detailed, enhanced and refined. Doing this the traceability links between elements have to be changed or extended. Newly added design elements have to be connected to analysis elements. Eventually, every analysis package has to be connected to one or more design subsystems, each analysis class has to be connected to one or more design classes and/or interfaces and each use case realization-analysis hast to be connected to a use case realization-design.

Refinement of Analysis Relations. During design the relations established between analysis objects have to be further refined and adopted to the chosen programming language. It is necessary to connect the original relation in the analysis model and the replacing elements in the design model by explicit traceability links of the type «refine». If an analysis class is realized in the design model by an attribute of a class or vice versa, this activity has to be documented by a traceability link as well.

Establishment of Subsystems and Components. The functional decomposition of the system into packages is started in the analysis phase and completed during the design phase. The parts of the system, separated by subsystems and their components communicate only using defined interfaces. Subsystems refining an analysis package are connected to this package by explicit traceability links of the type «refine». New introduced components and subsystems in the design model to fulfil non-functional requirements or constraints are connected by links of the type «realize».

Establishment of Use Case Realizations-Design. During analysis the use case realizations are used to answer the question, what the system has to do to realize a use case. During design these diagrams are further refined to show how it is to do. The design diagrams have to be connected by explicit traceability links of type «refine»with the corresponding diagram in the analysis model. Additionally established diagrams have to be connected by traceability links of the type «realize»with the related use case.

4.1.4 Activities of Implementation.

The design model is transformed into executable code during implementation. If it is possible to generate the source code automatically or a developer has to implement it, depends on the level of detail of the design

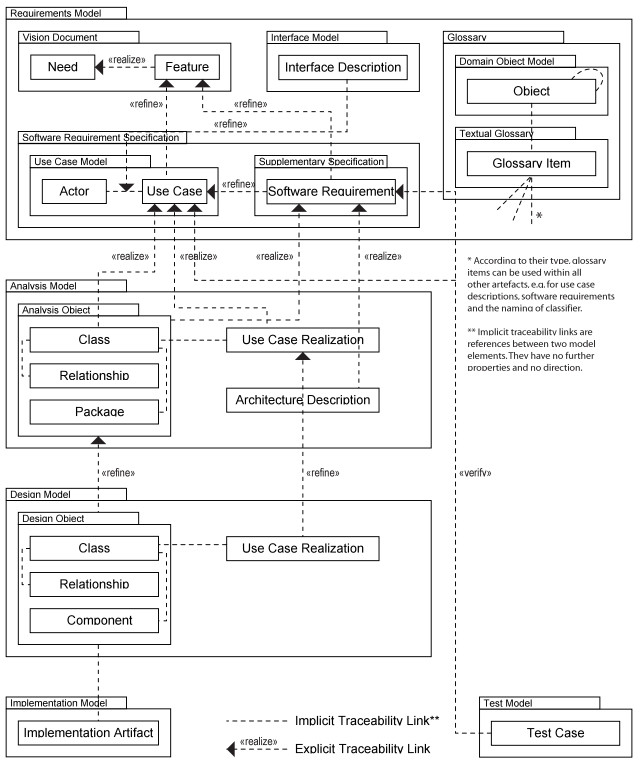

Figure 2. Traceability Link Model for the UP Workflows Requirements, OO-Analysis and Design

model. If the source code is generated automatically, no additional traceability is necessary. The used tool usually offers all functions necessary to follow a design object into implementation. If a developer is doing the transformation manually, it is possible to use implicit traceability by consistent naming of the implementation objects otherwise explicit traceability links have to be used. Traceability links are stored in the source code as annotations.

4.1.5 Flow Description by Activity Diagrams and State Machines.

Activity diagrams and state machines allow a modelling of processes without a prior definition of the structure of the system. Activity diagrams are especially used to describe flows, e.g. use cases, information flows between use cases (as interaction diagram) or methods and algorithms in the design model. State machines allow to model reactive objects, like classes, use cases, subsystems or whole systems. Both diagram types can be used in various situations within the development process, that's why they are discussed separately.

If an activity diagram or a state machine is used to describe a use case, a class or another model element, then both, the diagram and the model element have to be connected by an explicit traceability link of the type «refine». Alternatively, a consistent naming of the diagrams and the corresponding model element can be used for implicit traceability.

4.2. Verification of Traceability Links

The established traceability links have to be verified for completeness and correctness. This is necessary to ensure the usability and to avoid a decay of traceability information after changes to the connected models. In the following, rules for a validation are defined. Presently this set of rules as a first step covers the validation of the pure existence of traceability links. For reaching this aim the analysis of terms used in identifiers, the evaluation of relations in the class model or the analysis of use case descriptions is necessary. One example for a rule set introduced in Table 1 is: each use case has to be realized by at least one analysis class. This rule verifies the existence of at least one traceability link between both model elements. But it is not sufficient for the verification of correctness. Approaches for further validations offer the usage of terms in the model and the validation of plausibility between diagrams. For example, the analysis of terms means to search for glossary items of type object in the use case description and try to relate them to the identifier of the linked analysis classes and their attributes. Differences between both should lead to a notice for the developer.

Plausibility check between different diagrams means, that for each use case triggered by an actor, an analysis class of type interface has to be defined. Another case considering use case realizations, the classes of all instances within the use case realization have to be linked to the use case, because they realize it. In the following the so far known rules are listed:

Table 1. Traceability Link Verification Rules

Need ←«realize»– Feature (m:n)	
1. 2.	Each need is realized by at least one feature. Each feature is realizing at least one need.
Feature ←«refine»– Use Case (m:n)	
1. 2.	Each feature is refined by at least one use case. Each use case is refining at least one feature.
Use Case/Actor-Assoc. ←«refine»– Interf. Descript. (m:n)	
1. 2.	Each association between a use case and an actor is refined by at least one interface description. Each interface description is refining at least one association between use case actor.
Actor – – – – – Use Case (m:n)	
1. 2.	Each actor is associated to at least one use case. The associated actor(s) are the same as the actors used in the description of the use case.
Use Case ←«refine»– Suppl. Software Requirement (m:n)	
1.	Each software requirement (non-functional requirement, constraint) is refining at least one use case.
Use Case/Suppl. Softw. Req. ←«verify»– Test Case (m:n)	
1. 2.	Each software requirement is verified by at least one Test Case. Each Test Case is verifying at least one use case or software requirement.
Glossary – – – – – – DOM (1:0,1)	
1.	Each domain object is defined in the glossary.
Use Case ←«realize»– Analysis Class (m:n)	
1. 2.	Each use case is realized by at least one analysis class. Each analysis class is realizing at least one use case.
Use Case ←«realize»– Use Case Realization-Analysis (1:n)	
1.	Each use case realization is realizing one use case.

While applying the defined rules, one has to keep in mind that the UP is an incremental and iterative process. That means these rules will raise warnings as long as the model is not fully completed. However it is possible to check all chains of artefacts to the last existing artefact and all loose artefacts. An example for a loose artefact is a use case, which is realized by an analysis class, but does not refine any feature. This should lead to a warning for the developer.

The rule set is going to be expanded during the next steps of the project towards powerful support for developer.

5. Conclusions and Future Work

Traceability links improve the maintainability and support evolutionary development processes e.g., by recovering former development activities, especially for the case of changing requirements. In this paper the activities of a software process model have been enhanced by definitions and rules for traceability links to reduce the effort and to enable tool support. A model for traceability links has been introduced which can be tailored if necessary. Based on the development activities and artefacts, a set of rules for the verification of the traceability links has been developed.

As a part of our ongoing work, the developed traceability link model is currently completed and refined towards a complete coverage of the methodical activities, and to facilitate appropriate tool support for the creation, update and verification of the traceability links with a minimum interaction with the developer.

Other development methods and processes like Fusion [2] and Refactoring [3] are currently investigated aiming towards a broader applicability of the traceability model. To provide tool support, plug-ins for existing UML tools are currently developed. These plug-ins will support the developer by the establishment of traceability links in the background while modelling, and by maintaining the consistency of existing links during changes of artefacts.

Acknowledgments This work is partly funded by the German Research Foundation (Deutsche Forschungsgemeinschaft DFG) under id Ph49/7-1.

References

[1] J. Arlow and I. Neustadt. *UML 2 and the Unified Process Second Edition: Practical Object-Oriented Analysis and Design*. Addison-Wesley, 2005.

[2] D. Coleman. *Object-Oriented Development: The Fusion Method*. Prentice-Hall, 1994.

[3] M. Fowler. *Refactoring: Improving the Design of Existing Code*. Addison Wesley, 1999.

[4] I. Jacobson. *Object-Oriented Software Engineering: A Use Case Driven Approach*. Addison Wesley, Reading, Massachusetts, June 1992.

[5] P. Letelier. A framework for requirements traceability in UML-based projects. In *Proc. of 1st TEFSE*, Edinburgh, UK, Sept. 2002.

[6] P. Maeder, M. Riebisch, and I. Philippow. Traceability for managing evolutionary change. In *Proc. of 15th SEDE, Los Angeles, USA*, pages 1–8. ISCA, 2006.

[7] B. Ramesh and M. Jarke. Toward reference models of requirements traceability. *IEEE Trans. Software Eng.* 27(1):58–93, 2001.

[8] I. Spence and L. Probasco. Traceability strategies for managing requirements with use cases. Rational Software White Paper TP166, IBM, 2000.

[9] T. Weilkiens. *Systems Engineering mit SysML/UML*. dpunkt.verlag, 2006.