Customizing Traceability Links for the Unified Process

Patrick Maeder, Ilka Philippow and Matthias Riebisch

Software Systems/Process Informatics Group
Technical University of Ilmenau, Germany
patrick.maeder|ilka.philippow|matthias.riebisch@tu-ilmenau.de

Abstract. Traceability links are generally recognised as helpful means for improving the effectiveness of evolutionary development processes. However, their practical usage in analysis and design is still unsatisfying, especially due to the high effort required for creation, maintenance and verification of the links, and due to lacking or missing methods and tools for their management.
In this paper a concept for the systematic management of traceability is introduced, adapted for the and integrated into the Unified Process as one of the widely accepted software development methods. As an extension, requirements templates are applied to facilitate a tool supported analysis of natural language texts in use case descriptions. Template-based analyses enable a determination of types of terms and a check of their correct application as well as a recognition of implicit connections between development artefacts. A rule set is defined as a first step towards a powerful support of traceability handling. In the ongoing project the rule set is enhanced by heuristics and semantic-based rules to a whole framework of methods and rules.

1 Introduction

Complex, business critical software systems have to adapt to frequently changing needs. Evolutionary development processes have been developed to enable short responses to changes. In complex settings changes bear high risks, such as incomplete implementation, misunderstood dependencies, missing comprehension and lacking coverage. To manage these risks, the concept of traceability has been developed and introduced to the most development process standards. However, we have to state that traceability is poorly used in practice, and their usage is mostly limited to requirements engineering. Even in research, traceability is more discussed for requirements.

However, traceability links are needed in the areas of design and implementation as well. They facilitate design decisions and change impact analysis, they support program comprehension and they enable completeness checks for changes, if they can be maintained in a correct and complete state. For a comprehensive support of such design activities, traceability links have to be defined at a fine-grained level. Unfortunately, the maintenance of the links for such a way of design traceability requires an extremely high effort because a high number of links has to be managed, and many link maintenance tasks have to be

carried out manually. Two major open research questions have to be addressed: to master the amount and the complexity of traceability information, and to maintain and update the links. A tool support would be very helpful, but would require a traceability link update into development methods. Even if most design methods claim to support the concept of traceability, their definitions of artefacts, relations and activities are too imprecise to define traceability link update techniques.

As discussed earlier, our vision is the integration of traceability link management and maintenance into development methods and tools [1]. One of the challenges on this way consists in the refinement of the description of the major development methods. Detailed development activities are then extended by update activities for traceability links. The developer's activities are enriched as well, e.g. by describing the reasons and decisions for that activity. To meet the needs of the industrial practice it is necessary to perform this refinement for concrete development methods that are widely used in industry. In this paper we have chosen the Unified Process UP [2] for the definition of a process-specific model of traceability links. Although, Letelier showed in [3] the application of his metamodel for the UP, his definitions are not detailed enough to derive rules for traceability links. The UP description by its authors offers traceability as one of its features, but there is no detailed description of how and between which artefacts the traceability links should be established. Furthermore, works are necessary to define traceability links syntactically and semantically.

The contribution of this paper consists in an analysis and classification of 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. Additionally, a syntactic and semantic definition of traceability links is established customized to the UP's methods. This definition has been developed and validated in practical projects and case studies. These results constitute a milestone and provide a basis for further works towards our vision e.g., empirical investigations for rules concerning the suitable level of detail for traceability links, or for rules how far to follow traceability links during the impact analysis of a change.

The analysis of the UP and the customisation of the traceability concept are performed during practical development projects. As results of these works, guidelines for the level of detail and rules for the verification of traceability links have been established.

2 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 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.

The following concepts and definitions are based on the related works mentioned in section 4 as well as on our experiences from practical projects in the Automotive domain.

2.1 Categories of Traceability

Implicit Traceability. Implicit traceability results from existing associations between elements of the system model. For example, the use of the same identifier in an analysis and a design artefact expresses a dependency between both. The creation of this traceability link does not cause any additional effort.

Explicit Traceability. Explicit Traceability results from the establishing of connections between two artefacts during the software development process by a developer. It can be considered as an enhanced form of traceability [4], enabling the storage of additional information with the link. This information for example could be decisions made during analysis and design. By using explicit traceability links, program comprehension and changes of the system are facilitated.

The creation of explicit traceability requires additional effort of the developer. If one or both of two linked artefacts are changing, there is a risk that the traceability link is becoming inconsistent or invalid. It is necessary, to check explicit traceability links between changed artefacts, before they are used.

In contrast to explicit traceability implicit traceability describes references between two model elements, without any additional properties. It is possible to search for implicit connections, to store them and make them explicit. Thus, the benefits of adding additional information are given. But, in this case it is necessary to verify the correctness of the link, before using it.

2.2 Traceability Links

Components of Traceability Links. In the following we define the components of an explicit traceability link. This definition is driven by the goal of (semi)automatic support for link establishment and maintenance and focuses on getting the highest possible benefit from the usage of traceability. It provides the required data e.g. for the conservation of design decisions and for performing an impact analysis. This definition was established based on an analysis of the related works mentioned in section 4 and by our experiences from projects in the Automotive domain. An explicit traceability link consists of:

- a unique identifier for its recognition and to avoid ambiguity,
- a start element as source of the link, including type and context of this element (e.g. a class of the analysis model)
- an end element as destination of the link, including type and context of this element
- the type of the link
- the development decision connected with the link, including the goal of the decision, alternatives, rating of the alternatives and the choice

The link can contain additional information:

– the link status concerning the certainty of correctness (e.g. after changes of one or both of the connected elements),
– the creator of the link and
– a priority, which shows the importance of the link and allows to check only high prioritised links after changes of elements (according to [5]).

A traceability link is syntactical defined in Backus-Naur-form as follows: This

```
Traceability Link::= <ID> <Start element> <End element> <Type> <Decision>
                     [ <Status> ] [ <Developer> ] [ <Priority> ]
Start element::=     <ID>
End element::=       <ID>
Type::=              refine | realize | verify | define
Decision::=          <Goal> <Alternatives> <Choice>
Status ::=           0 | . . . | 100 "%"
Developer::=         <Text>
Priority::=          0 | 1
Goal ::=             <Text>
Alternatives::=      <Alternative> | <Alternative> <Alternatives>
Choice::=            <Alternative.ID>
Alternative ::=      <Alternative.ID> <Text>
Alternative.ID::=    <Number>
```

definition of a link provides all information required for the link establishment and update as well as for the traceability goals mentioned above. It conforms to the UML metamodel [2].

Types of Traceability Links. The traceability link type shows the relationship between two connected elements and/or the development activity for the generation of the destination element from the source element. A reduction of the number of types of traceability links aims at a minimization of the necessary number of rules for establishing and checking of links. Several authors use different types of links for different concepts: The link types of UML [2] and its extension SysML [6], by Letelier [3] and in the link metamodel of Ramesh and Jarke [7] differ in its concepts and categorisation. Based on the analysis of theses works of related works (sect. 4) and on our experience from projects and case studies, the following four basic types of traceability links have been identified:

– 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).

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.

3 Software Development Processes and Methods

Software development processes consist of activities and artefacts leading from requirements to the systems implementation. The handling of traceability links can be the more automated 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 fine-grained the process description is the easier is the defining of rules for creation and updating of traceability links. Therefore, the approach for traceability proposed in this paper is presently focused on one concrete process, the Unified Process.

3.1 The Unified Process UP

In the UP several ancestor methods like Object-Oriented Software Engineering OOSE [8] 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 [2]. For establishing traceability links especially the requirements activities of the UP have to be more detailed and enhanced. For this purpose text templates akin to those in [9] are integrated into the process.

3.2 Describing Requirements by Text-Templates

Chris Rupp et al. describe in [9] the requirements development as a three-step process, consisting of formulating, analysing and successive improving requirements by rules. A concept of so-called requirements patterns is introduced to accelerate this process. Rupp characterises it as a general concept to construct natural language requirements based on formal defined elements, which are verifiable and can be modelled. A pattern consists basically of one or more generic, syntactic requirements templates. Furthermore it consists of a semantic definition of important parts of the template, of logical operators to combine conditions and of rules to define test criteria's. Using the templates supports the definition of test cases and the identification of objects.

In [9] Rupp et al. categorize requirements into three types: independent system activity, user interaction and interface requirement. Figure 1 shows the elements of all three types of requirements combined into one graphic. Each requirement is based on a functionality, which is described by a so-called process word. A process word is strictly a verb defined in a process word list. From top to bottom the grey shaded boxes correspond to the introduced three types. If the requirement is of type user interaction an actor has to be filled into the template. The legal classification can be chosen by using one of the words: shall, should or

will. To complete the requirements expression an object and its enhancements and optional logical and time constraints have to filled into the template.

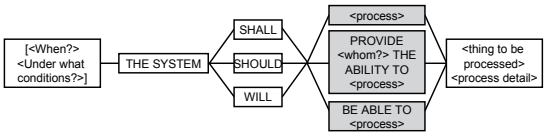

Fig. 1. Requirements Templates for all Three Kinds of Requirements

In addition to the advantages of well formulated, verifiable requirements, the usage of requirements templates offers some more benefits. Requirements templates support traceability. Rules can be defined based on the type of the requirement, its elements and the position of these elements within the sentence. Eventually this supports the consistency between different model elements. A detailed explanation of this support is given in the next chapter.

The Rupp text templates can effectively support the structured description of use cases according to [10]. For enabling the definition of activities in use case descriptions we have coupled the text templates closely to the development glossary: all terms filled into the template have to be defined in the glossary. The actors of the user activity have to be the same as these specified in the field actors of the use case description.

User Activity <actor> <process> <thing to be processed and details>.
System Activity The System [shall/should/will]
<thing to be processed and details> <process>.

3.3 Traceability Relevant Artefacts in the UP

In this section traceability relevant artefacts of the UP are introduced. We focus mainly on the workflows requirements and analysis/design, as these contain the traceability relevant activities and artefacts.

Artefacts of the Requirements Workflow.

Requirements. Requirements describe properties or features of the system that has to be developed. In the UP they are not hierarchically ordered instead they represent different views for different groups of stakeholders.

The Vision Document contains Needs, describing informally what the stakeholders expect of the system and Features, describing informally what the system offers to fulfil these needs.

The Software Requirements Specification (SRS) consists of Software Requirements, which are commonly divided in functional and non-functional requirements and constraints. The UP centralizes these requirements in two artefacts:

the Use Case Model, consisting of all functional requirements and the Supplementary Specification, consisting of all non-functional requirements and constraints expressed as declarative statements.

Glossary. The glossary lists terms of the project domain and gives a definition to each of them. The strict usage of defined glossary terms in all development phases enables automated generation of connections between the same terms used in different artefacts. Every special term from the very beginning of a project has to be defined in the glossary. Only defined terms are allowed to be used during the development process. That means, that the identifier of all model elements consist only of defined terms. Also the elements of the before introduced requirements templates have to be defined in the glossary. Glossary items can be categorised into type groups, according to Rupp [9] in three types: actor, object and process. By using additional information about the type of a term, rules can be identified for suggesting a special term while naming an object or writing a requirement. These rules can also be used to support the verification of the right usage of terms in the model.

Domain Object Model (DOM). The DOM represents glossary items as classes in UML class diagrams. The usage of equal names in both artefacts realizes a connection as implicit traceability links.

Interface Description. Interfaces are described, depending on their kind as e.g., prototypes of graphical user interfaces, drawings or textual descriptions.

Artefacts of the Object-Oriented Analysis.

Analysis Class. Analysis classes define the necessary structure for the realization of a use case in the system. Class identifier must be meaningful and domain specific and have to be defined in the glossary.

Package. Packages organize model elements and diagrams in groups.

Use Case Realization-Analysis. Use case realizations consist of a set of diagrams describing a use case specification. For visualisation of the structure a class diagram is used. Interaction diagrams describe the communication between these classes.

Relation Between Analysis Objects. Relations visualize functional or structural dependencies. The following relations can occur between analysis objects: Association, Generalization, Dependency and Hierarchy.

Analysis Model. An analysis model consists of all artefacts, developed during the analysis workflow.

Architectural Description. An architecture description is a short textual summary of architecture relevant aspects of the system.

Artefacts of the Object-Oriented Design.

Design Class. Design classes are refined and detailed classes, suitable and ready for implementation.

Use Case Realization-Design. Use case realizations-design describe the collaboration of several design objects for use case realization.

Subsystem and Component. Subsystems and components result from decomposing complex systems into smaller, easier manageable parts of the system.

Design Model. A design model is a refinement of the analysis model and is enhanced with more details and particular technical solutions. The elements of the design model have to be specified as far, that they can be implemented.

3.4 Development Activities and Relations Between Model Elements

In this section a model of useful traceability links for the UP is proposed. At first the UP development activity is named and then, related to it, traceability links between the developed artefacts are introduced. For illustration, every step is explained by a simple example, for a wiper control. The activity chart in Fig. 2 contains only those activities necessary for the establishment of traceability links. 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.

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 «

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.

In our example three terms have been identified based on the feature definition: wiping speed, interval time and single wipe. They have to be defined and listed in the glossary. These terms and those for the next development activities are defined in Table 1. realize»(see Fig. 3).

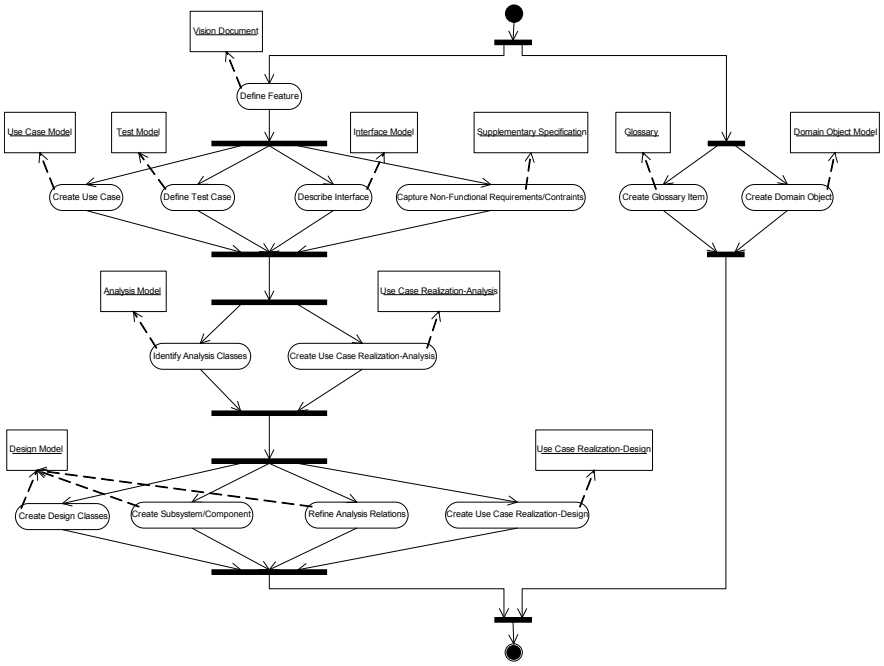

Fig. 2. Development Activities of the Unified Process Workflows: Requirements and Analysis/Design

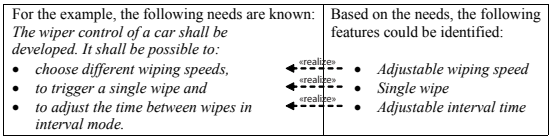

Fig. 3. Traceability Links between Needs and Features for the Wiper Control Example

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 (Table 1). 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

Table 1. Glossary of the Wiper Control Example

	Term	Definition	Type
1	Driver	Person who drives a car	Actor
2	Clamp 15	An electrical connection, which is getting active when ignition is switched on.	Object
3	Steering Column Switch	Switch to choose the wiping speed with the positions off, interval, slow and fast.	Object
4	Wiping Speed	Speed of the wiping blade	Object
5	Interval Time	Time between to wipings in interval mode	Object
6	Wiping	Moving the wiper blade from its start position to its end position and back.	Object
7	Single Wipe	Manually triggered single wiping	Object
8	Choose	The user selects one or more elements from a finite set of elements.	Process
9	Set	The system logically chooses the value of a certain figure, according to selection criteria from a finite set.	Process
10	Trigger	The user starts by a certain action a process of the system.	Process

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. For the description of use cases, text templates akin to Rupp [9] are used (see section 3.2).

In the example the following three use cases have been identified: Set Wiping Speed, Choose Interval Time, and Trigger Instant Wipe. These use cases are connected to the before defined features by explicit traceability links of type «refine»(see Fig. 4).

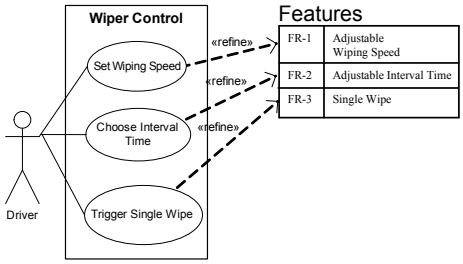

Fig. 4. Traceability Links between Features and Use Cases for the Wiper Example

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». In the example there is only one interface between driver and system, the steering column switch of the car (see Fig. 5).

Table 2. Example Description of the Use Case "Set Interval Time"

Name	Choose Interval Time
Description	This use case allows the driver to set a new interval time, which is waited between two wipes.
Actors	Driver
Rationale	Steering column switch has been set to position interval.
Precondition	Clamp 15 is active and the steering column switch has position OFF.
Normal Flow 1	The driver switches the steering column switch to position INTERVALL.
2	The driver switches the steering column switch to position OFF.
3	The driver switches the steering column switch after not more than 30s to position INTERVALL.
4	The system has to set the new interval time as the time the steering column switch has been in position OFF.
Altern. Flow	no
Postconditions	no

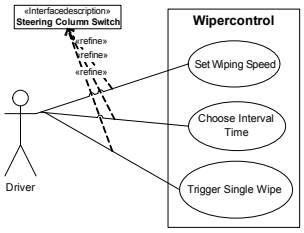

Fig. 5. Example of an Interface Description

Development Activities of Object-Oriented 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.

In the example three analysis classes are defined for the three use cases (see Fig. 6). All use cases are triggered by the driver using the same interface (see Fig. 5. Therefore all use cases are connected by traceability links with the interface class Steering Column Switch. The use cases Set Wiping Speed and Trigger Instant Wipe are realized by the class Wiper Control and the use case Choose Interval Time is realized by the class Interval Reader. All these development activities are traceable through the corresponding links.

Performing of Use Case Realizations-Analysis. In this step the cooperation between the different analysis classes has to be described by UML interaction

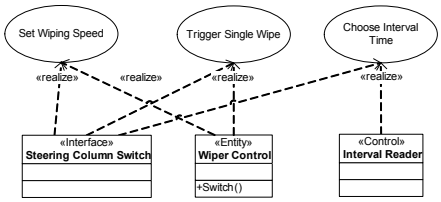

Fig. 6. Identified Analysis Classes to the Wiper System

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 implicitly 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. The sequence chart in Fig. 7 specifies the necessary communication between the analysis classes, to realize the use case Choose Interval Time from the example.

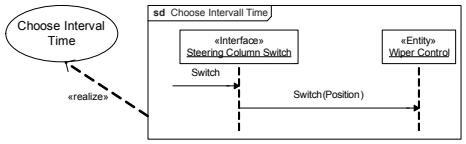

Fig. 7. Example of a Use Case Realization

Development Activities during Design.

Creation of Design Classes (Design Class 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 step by step. 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. The replacement e.g. of a bidirectional association by two unidirectional associations is shown in Fig. 8.

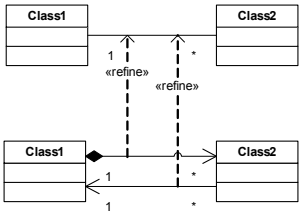

Fig. 8. Refining Analysis Relations to be able to Implement Them

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». Newly introduced components and subsystems in the design model to fulfil non-functional requirements or constraints are connected by traceability 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.

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 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.

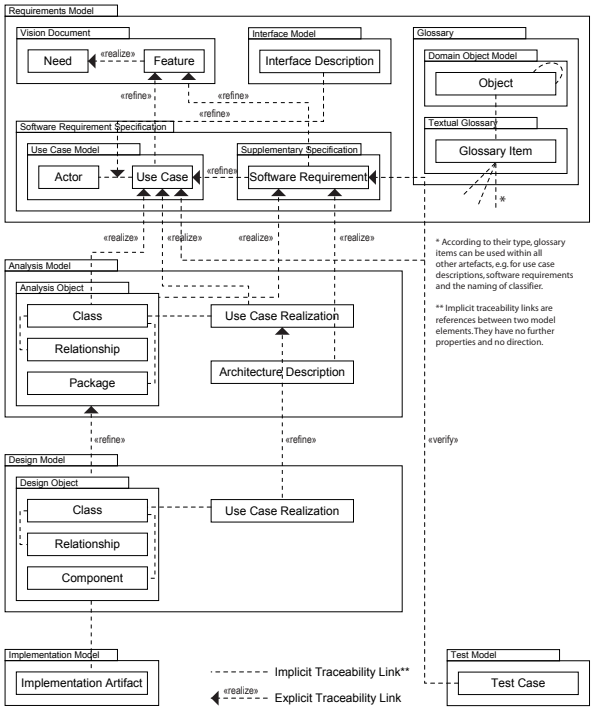

Fig. 9. Traceability Links between Artefacts of Requirements Analysis, Object-Oriented Analysis and Design

Flow Description by Activity Diagrams and State Machines. Activity diagrams and state machines allow the modelling of processes without a predefined 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.

3.5 Tailoring of the Traceability Model

The introduced traceability links are summarized in Fig. 9. The presented traceability model has been developed based on experiences, exploration of the UP and best practices in software engineering. The traceability links allow following

essential development activities. However, tailoring is possible and sometimes necessary depending on the complexity of the project, the expected results by using traceability and the available resources to establish traceability. There are two possible ways to tailor the traceability model to meet special needs:

1. omitting or adding traceability connections of the model and
2. enhancing or decreasing the level of granularity of defined links.

The first point is reasonable in the case that at the same time corresponding development activities are omitted or added. Examples for such scenarios are:

- software development without object-oriented analysis for very small and short-living projects or
- requirements analysis without feature definition.

The second point, the change of the level of granularity, refers in particular to traceability links between use cases and analysis objects and between analysis and design objects. Here a high number of traceability links has to be established, but at the same time a higher level of granularity can support the developer with valuable information, e.g. if traceability links between use cases and analysis classes are used. It is possible to connect artefacts with a higher or lower level of detail at both sides of the traceability link, then described in our model before. In the case of use cases, more detail means to link parts of the use case descriptions and less detail means to link to features instead of use cases. For analysis classes more detail means to connect to methods and attributes and less detail means to connect to the package, which contains the class. Figure 10 shows three examples of possible traceability links. The level of detail is rising from left to right.

It has to be pointed out, that while using a higher level of detail, the traceability links of fewer detail are included implicitly. That means by connecting a use case action with an operation of an analysis class, there is also an implicit connection between use case and class.

In principle it is possible to increase or lower the level of detail only at one connection side, but without real advantage because the resulting information is not more precise. An advantage is only reached by a corresponding change on both sides of the connection.

3.6 Verification of Traceability Links

Defined traceability links have to be verified for completeness and correctness. Only thus the usability can be assured and a decay of traceability information after changes of the connected models can be avoided. In the following rules for validation are defined. Presently this set of rules is a first step for validating only 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. E.g. one of the rule set introduced in the list below 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

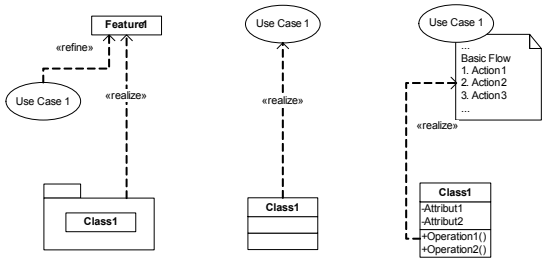

Fig. 10. Example of the Level of Traceability between Use Cases and Analysis Objects

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 Table 3 the so far known rules are listed.

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.

4 Related Work

A comprehensive description of research topics, results and open issues in the field of traceability was given in a former publication [11]. In this paper according to the specific topic, three studies concerning traceability frameworks has to be pointed out in particular.

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 explain a simplified and a full version of their metamodel. Further they give a predefined standard set of link types. The authors focus especially on project management and organizational needs of traceability. They do not give answers to the problem how traceability should be established in analysis and design.

Table 3. Verification Rules for Traceability Links

Need ←«realize»– Feature (m:n)	
1.	Each need is realized by at least one feature.
2.	Each feature is realizing at least one need.
Feature ←«refine»– Use Case (m:n)	
1.	Each feature is refined by at least one use case.
2.	Each use case is refining at least one feature.
Use Case/Actor-Assoc. ←«refine»– Interf. Descript. (m:n)	
1.	Each association between a use case and an actor is refined by at least one interface description.
2.	Each interface description is refining at least one association between use case actor.
Actor – – – – – Use Case (m:n)	
1.	Each actor is associated to at least one use case.
2.	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.	Each software requirement is verified by at least one Test Case.
2.	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.	Each use case is realized by at least one analysis class.
2.	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.

Spence and Probasco [4] discuss several alternatives for traceability between requirements. The paper is focused on the UP. They do not give answers to the question how the transition to analysis and design and the on-going work should be traced.

Letelier [3] 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 general useable, it is not possible to define rules and activities for the creation, verification and the update of links, which could be carried out (semi)automatically by a tool.

5 Conclusions and Future Work

In this paper the general activities of a software process model have been enhanced by the establishment of traceability links to reduce the effort and to enable tool support. Traceability links improve the maintainability and support evolutionary development processes e.g., by recovering former development activities, especially for the case of changing requirements. 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 ongoing work, the developed traceability link model is currently completed and refined towards a complete coverage of the methodical activities,

to facilitate appropriate tool support for the creation, update and verification of the traceability links with a minimum interaction with the developer. For refining the model, architectural development methods like Qasar [12] are investigated and integrated.

Other development methods and processes like Fusion [13] and Refactoring [14] are currently investigated aiming towards a generally usable traceability model. For the realization of tool support we have started the implementation of plug-ins for existing UML tools. The 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. However, a consequent application of the rules of the development method in all modelling activities constitutes a precondition for such a support.

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

References

1. Riebisch, M.: Supporting evolutionary development by feature models and traceability links. In: Proceedings 11th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS2004), Brno, Czech Republic (May 2004) 370–377
2. Arlow, J., Neustadt, I.: UML 2 and the Unified Process Second Edition: Practical Object-Oriented Analysis and Design. Addison-Wesley (2005)
3. Letelier, P.: A framework for requirements traceability in UML-based projects. In: Proceedings of 1st International Workshop on Traceability in Emerging Forms of Software Engineering, Edinburgh, UK (September 2002)
4. Spence, I., Probasco, L.: Traceability strategies for managing requirements with use cases. Rational Software White Paper TP166, IBM (2000) `http://www-306.ibm.com/software/rational/info/literature/whitepapers.jsp`.
5. Cleland-Huang, J., Chang, C.K., Christensen, M.J.: Event-based traceability for managing evolutionary change. IEEE Trans. Software Eng **29**(9) (2003) 796–810
6. Weilkiens, T.: Systems Engineering mit SysML/UML. dpunkt.verlag (2006)
7. Ramesh, B., Jarke, M.: Toward reference models of requirements traceability. IEEE Trans. Software Eng **27**(1) (2001) 58–93
8. Jacobson, I.: Object-Oriented Software Engineering: A Use Case Driven Approach. Addison Wesley, Reading, Massachusetts (June 1992)
9. Rupp, C., et al.: Requirements-Engineering und Management. Carl Hanser Verlag (2007)
10. Cockburn, A.: Using goal-based use cases. JOOP **10**(7) (1997) 56–62
11. Maeder, P., Riebisch, M., Philippow, I.: Traceability for managing evolutionary change. In: Proceedings of 15th International Conference on Software Engineering and Data Engineering, Los Angeles, USA, ISCA (2006) 1–8
12. Bosch, J.: Design and Use of Software Architectures : Adopting and evolving a product-line approach. Addison-Wesley (2000)
13. Coleman, D.: Object-Oriented Development: The Fusion Method. Prentice-Hall (1994)
14. Fowler, M.: Refactoring: Improving the Design of Existing Code. Addison Wesley (1999)