Traceability Handling in Model-based Prediction of System Quality

Aida Omerovic*† and Ketil Stølen*†
**SINTEF ICT, Pb. 124, 0314 Oslo, Norway*
†*University of Oslo, Department of Informatics, Pb. 1080, 0316 Oslo, Norway*
Email: {aida.omerovic,ketil.stolen}@sintef.no

***Abstract*—Our earlier research indicated the feasibility of the PREDIQT method for model-based prediction of impacts of architectural design changes, on the different quality characteristics of a system. The PREDIQT method develops and makes use of a multi-layer model structure, called prediction models. Usefulness of the prediction models requires a structured documentation of both the relations between the prediction models and the rationale and assumptions made during the model development. This structured documentation is what we refer to as trace-link information. In this paper, we propose a traceability scheme for PREDIQT, and an implementation of it in the form of a prototype tool which can be used to define, document, search for and represent the trace-links needed. The solution is applied on prediction models from an earlier PREDIQT-based analysis of a real-life system. Based on a set of success criteria, we argue that our traceability approach is useful and practically scalable in the PREDIQT context.**

***Keywords*-traceability; system quality prediction; modeling; architectural design; change impact analysis; simulation.**

I. Introduction

We have developed and tried out the PREDIQT method [1] [2] aimed for predicting impacts of architectural design changes on system quality characteristics and their trade-offs. Examples of quality characteristics include availability, scalability, security and reliability.

Important preconditions for model-based prediction are correctness and proper usage of the prediction models. The process of the PREDIQT method guides the development and use of the prediction models, but the correctness of the prediction models and the way they are applied are also highly dependent on the creative effort of the analyst and his/her helpers. In order to provide additional help and guidance to the analyst, we propose in this paper a traceability approach for documenting and retrieving the rationale and assumptions made during the model development, as well as the dependencies between the elements of the prediction models.

The approach is defined by a traceability scheme, which is basically a feature diagram specifying capabilities of the solution and a meta-model for the trace-link information. A prototype tool is implemented in the form of a relational database with user interfaces which can be employed to define, document, search for and represent the trace-links needed. The solution is illustrated on prediction models from an earlier PREDIQT-based analysis conducted on a real-life system [3].

The paper is organized as follows: Section II provides background on traceability. The challenge of traceability handling in the context of the PREDIQT method is characterized in Section III. Our traceability handling approach is presented in Section IV. Section V illustrates the approach on an example. Section VI argues for completeness and practicability of the approach, by evaluating it with respect to the success criteria. Section VII substantiates why our approach, given the success criteria outlined in Section III, is preferred among the alternative traceability approaches. The concluding remarks and future work are presented in Section VIII.

A full technical report [4] is available and includes: 1) an outline of the PREDIQT method, 2) guidelines for application of the prediction models which the success criteria for our traceability approach are deduced from, and 3) further details on traceability in PREDIQT.

II. Background on traceability

IEEE [5] provides two definitions of traceability:

1) Traceability is the degree to which a relationship can be established between two or more products of the development process, especially products having a predecessor-successor or master-subordinate relationship to one another; for example, the degree to which the requirements and design of a given software component match.
2) Traceability is the degree to which each element in a software development product establishes its reason for existing.

Traceability research and practice are most established in fields such as requirements engineering and model-driven engineering (MDE). Knethen and Paech [6] argue: "Dependency analysis approaches provide a fine-grained impact analysis but can not be applied to determine the impact of a required change on the overall software system. An imprecise impact analysis results in an imprecise estimate of costs and increases the effort that is necessary to implement a required change because precise relationships have to be identified during changing. This is cost intensive and error prone because analyzing the software documents requires detailed understanding of the software documents and the relationships between them." Aizenbud-Reshef et al. [7] furthermore state: "The extent of traceability practice is viewed as a measure of system quality and process maturity and is mandated by many standards" and "With complete traceability, more accurate costs and schedules of changes can be determined, rather than depending on the programmer to know all the areas that will be affected by these changes".

IEEE [5] defines a trace as "A relationship between two or more products of the development process." According to the OED [8], however, a trace is defined more generally as a "(possibly) non-material indication or evidence showing what has existed or happened". As argued by [9]: "If a developer works on an artifact, he leaves traces. The software configuration management system records who has worked on the artifact, when that person has worked on it, and some systems also record which parts of the artifacts have been changed. But beyond this basic information, the changes themselves also reflect the developer's thoughts and ideas, the thoughts and ideas of other stakeholders he may have talked to, information contained in other artifacts, and the transformation process that produced the artifact out of these inputs. These influences can also be considered as traces, even though they are usually not recorded by software configuration management systems."

A traceability link is a relation that is used to interrelate artifacts (e.g., by causality, content, etc.) [9]. In the context of requirements traceability, [9] argues that "a trace can in part be documented as a set of meta-data of an artifact (such as creation and modification dates, creator, modifier, and version history), and in part as relationships documenting the influence of a set of stakeholders and artifacts on an artifact. Particularly those relationships are a vital concept of traceability, and they are often referred to as traceability links. Traceability links document the various dependencies, influences, causalities, etc. that exist between the artifacts. A traceability link can be unidirectional (such as depends-on) or bidirectional (such as alternative-for). The direction of a link, however, only serves as an indication of order in time or causality. It does not constrain its (technical) navigability, so traceability links can always be followed in both directions".

In addition to the different definitions, there is no commonly agreed basic classification [9]. A taxonomy of the main concepts within traceability is suggested by [6].

An overview of the current state of traceability research and practice in requirements engineering and model-driven development is provided by [9], based on an extensive literature survey. Another survey [10] discusses the state-of-the-art in traceability approaches in MDE and assesses them with respect to five evaluation criteria: representation, mapping, scalability, change impact analysis and tool support. Moreover, Spanoudakis and Zisman [11] present a roadmap of research and practices related to software traceability.

Traces can exist between both model- and non-model artifacts. The means and measures applied for obtaining traceability are defined by so-called traceability schemes. A traceability scheme is driven by the planned use of the traces. The traceability scheme determines for which artifacts and up to which level of detail traces can be recorded [9]. A traceability scheme thus defines the constraints needed to guide the recording of traces, and answers the core questions: what, who, where, how, when and why. Additionally, there is tacit knowledge (such as why), which is difficult to capture and to document. A traceability scheme helps in this process of recording traces and making them persistent.

According to Wieringa [12], representations and visualizations of traces can be categorized into matrices, cross-references, and graph-based representations. As elaborated by Wieringa, the links, the content of the one artifact, and other information associated with a cross reference, is usually displayed at the same time. This is however not the case with traceability matrices. So, compared to traceability matrices, the user is (in the case of cross-references) shown more local information at the cost of being shown fewer (global) links. As models are the central element in MDE, graph-based representations are the norm. A graph can be transformed to a cross-reference. Regarding the notation, there is, however, no common agreement or standard, mostly because the variety and informality of different artifacts is not suitable for a simple, yet precise notation.

Traceability activities are generally not dependent on any particular software process model. Knethen and Paech [6] argue that the existing traceability approaches do not give much process support. They specify four steps of traceability process: 1) define entities and relationships, 2) capture traces, 3) extract and represent traces, and 4) maintain traces. Similarly, Winkler and Pilgrim [9] state that traceability and its supporting activities are currently not standardized. They classify the activities when working with traces into: 1) planning for traceability, 2) recording traces, 3) using traces, and 4) maintaining traces.

Trace models are usually stored as separate models, and links to the elements are (technically) unidirectional in order to keep the connected models or artifacts independent. Alternatively, models can contain the trace-links themselves and links can be defined as bidirectional. While embedded trace-links pollute the models, navigation is much easier [9]. Thus, we distinguish between external and internal storage, respectively. Anquetil at al. [13] argue: "Keeping link information separated from the artifacts is clearly better; however it needs to identify uniquely each artifact, even fined-grained artifacts. Much of the recent research has focused on finding means to automate the creation and maintenance of trace information. Text mining, information retrieval and analysis of trace links techniques have been successfully applied. An important challenge is to maintain links consistency while artifacts are evolving. In this case, the main difficulty comes from the manually created links, but scalability of automatic solution is also an issue."

Various tools are used to set and maintain traces. Surveys of the tools available are provided by [6], [9], [11] and [7]. Bohner and Arnold [14] found that the granularity of documentation entities managed by current traceability tools is typically somewhat coarse for an accurate impact analysis.

III. The challenge

Three interrelated sets of models are developed during the process of the PREDIQT method: Design Model which specifies system architecture, Quality Model which specifies the system quality notions, and Dependency Views (DVs) which represent the interrelationship between the system quality and the architectural design. The PREDIQT process consists of three overall phases: *Target modeling*, *Verification of prediction models*, and *Application of prediction models*.

Figure 1 provides an overview of the elements of the prediction models, expressed as a UML [15] class diagram. A Quality Model is a set of tree-like structures which clearly specify the system-relevant quality notions, by defining and decomposing the meaning of the system-relevant quality terminology. Each tree is dedicated to a target system-relevant quality characteristic. Each quality characteristic may be decomposed into quality sub-characteristics, which in turn may be decomposed into a set of quality indicators. As indicated by the relationship of type aggregation, specific sub-characteristics and indicators can appear in several Quality Model trees dedicated to the different quality characteristics. Each element of a Quality Model is assigned a quantitative normalized metric and an interpretation (qualitative meaning of the element), both specific for the target system. A Design Model represents the relevant aspects of the system architecture, such as for example process, dataflow, structure and rules.

A DV is a weighted dependency tree dedicated to a specific quality characteristic defined through the Quality Model. As indicated by the attributes of the Class *Node*, the nodes of a DV are assigned a name and a QCF (*Quality Characteristic Fulfillment*). A QCF is value of the degree of fulfillment of the quality characteristic, with respect to what is represented by the node. The degree of fulfillment is defined by the metric (of the quality characteristic) provided in the Quality Model. Thus, a complete prediction model has as many DVs as the quality characteristics defined in the Quality Model. Additionally, as indicated by the *Semantic* dependency relationship, semantics of both the structure and the weights of a DV are given by the definitions of the quality characteristics, as specified in the Quality Model. A DV node may be based on a Design Model element, as indicated by the *Based on* dependency relationship. As indicated by the self-reference on the *Node* class, one node may be decomposed into children nodes. Directed arcs express dependency with respect to quality characteristic by relating each parent node to its immediate children nodes, thus forming a tree structure. Each arc in a DV is assigned an EI (*Estimated Impact*), which is a normalized value of degree of dependence of a parent node, on the immediate child node. The values on the nodes and the arcs are referred to as parameter estimates. We distinguish between prior and inferred parameter estimates. The former ones are, in the form of empirical input, provided on leaf nodes and all arcs, while the latter ones are deduced using the DV propagation model for PREDIQT [3].

The intended application of the prediction models does not assume implementation of change on the target system, but only simulation of effects of the independent architectural design changes quality of the system (in its currently modelled state). Since the simulation is only performed on the target system in its current state and the changes are simulated independently (rather than incrementally), versioning of the prediction models in not necessary. Hence, maintenance of both prediction models and trace information is beyond the scope of PREDIQT.

Trace-link information can be overly detailed and extensive while the solution needed in a PREDIQT context has to be applicable in a practical real-life setting within the limited resources allocated for a PREDIQT-based analysis. Therefore, the traceability approach should provide sufficient breadth and accuracy for documenting, retrieving and representing of the trace-links, while at the same time being practically applicable in terms of comprehensibility and scalability. The right balance between the completeness and accuracy of the trace information on the one side, and practical usability of the approach on the other side, is what characterizes the main challenge in proposing the appropriate solution for traceability handling in PREDIQT. Therefore, the trace-link creation efforts have to be concentrated on the traces necessary during the application of the prediction models.

It is, as argued by [9], an open issue to match trace usage and traceability schemes, and to provide guidance to limit and fit traceability schemes in a such way that they match a projects required usage scenarios for traces. One of the most urgent questions is which requirements a single scenario imposes on the other activities (in particular planning and recording) in the traceability process.

Moreover, it is argued by Aizenbud-Reshef et al. [7] that the lack of guidance as to what link information should be produced and the fact that those who use traceability are commonly not those producing it, also diminishes the motivation of those who create and maintain traceability information. In order to avoid this trap, we used the PREDIQT guidelines [4] for the analyst as a starting point, for deriving the specific needs for traceability support. The guidelines are based on the authors' experiences from industrial trials of PREDIQT [3] [2]. As such, the guidelines are not exhaustive but serve as an aid towards a more structured process of applying the prediction models and accommodating the trace

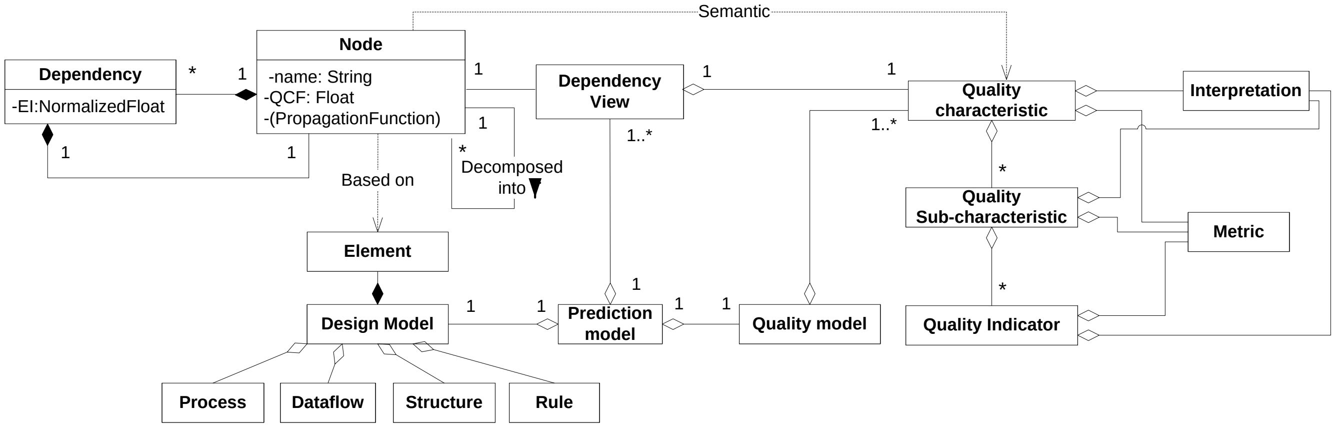


Figure 1. An overview of the elements of the prediction models, expressed as a UML class diagram

information during the model development, based on the needs of the "Application of prediction models"-phase.

The specific needs for traceability support in PREDIQT are summarized below:

1) There is need for the following kinds of trace-links:
   - links between the Design Model elements
   - links from the Design Model elements to DV elements
   - links from DV elements to Quality Model elements (i.e. traces to the relevant quality indicators and rationale for the prior estimates)
   - links to external information sources (documents, measurement, domain experts) used during the development of DV structure and estimation of the parameters
   - links to rationale and assumptions for: Design Model elements, the semantics of the DV elements, as well as structure and prior parameter estimates of the DVs
2) The traceability approach should have facilities for both searching with model types and model elements as input parameters, as well as for reporting linked elements and the link properties
3) The traceability approach should be flexible with respect to granularity of trace information
4) The traceability approach should be practically applicable on real-life applications of PREDIQT

These needs are in the sequel referred to as the **success criteria** for the traceability approach in PREDIQT.

IV. Our solution

This section starts by presenting our traceability scheme for PREDIQT. Then, a prototype tool for trace-link management, implementing the needs specified through the traceability scheme, is presented.

A. Traceability scheme

We propose a traceability scheme in the form of a meta-model for trace-link information and a feature diagram for capabilities of the solution. The types of the trace-links and the types of the traceable elements are directly extracted from Success Criterion 1 and represented through

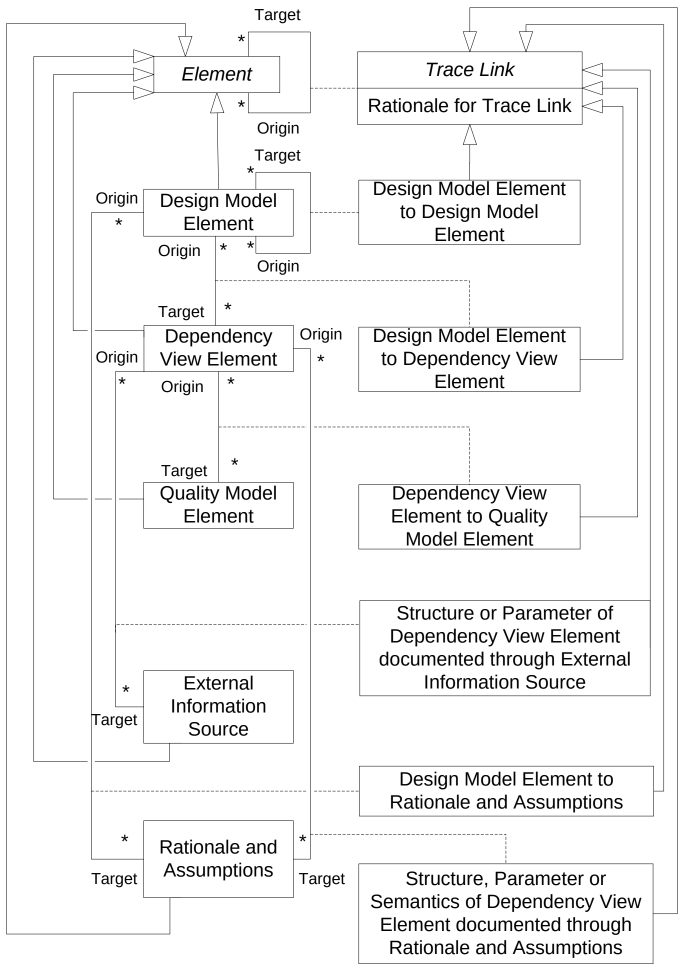

Element
Trace Link
Rationale for Trace Link
Target
Origin
Design Model Element
Design Model Element to Design Model Element
Dependency View Element
Design Model Element to Dependency View Element
Quality Model Element
Dependency View Element to Quality Model Element
Structure or Parameter of Dependency View Element documented through External Information Source
External Information Source
Design Model Element to Rationale and Assumptions
Rationale and Assumptions
Structure, Parameter or Semantics of Dependency View Element documented through Rationale and Assumptions

Figure 2. A meta model for trace-link information, expressed as a UML class diagram

a meta-model shown by Figure 2. The *Element* abstract class represents a generalization of a traceable element. The *Element* abstract class is specialized into the five kinds of traceable elements: *Design Model Element*, *DV Element*, *Quality Model Element*, *External Information Source*, and *Rationale and Assumptions*. Similarly, the *Trace Link* abstract class represents a generalization of a trace-link and may be assigned a rationale for the trace-link. The *Trace Link* abstract class is specialized into the six kinds of trace-links.

Pairs of certain kinds of traceable elements form binary relations in the form of unidirectional trace-links. Such relations are represented by the UML-specific notations called association classes (a class connected by a dotted line to a link which connects two classes). For example, trace-links of type *Design Model Element to Design Model Element* may be formed from a *Design Model Element* to a *Dependency View Element*. The direction of the link is annotated by the origin (the traceable element that the trace-link goes from) and the target (the traceable element that the trace-link goes to). Since only distinct pairs (single instances) of the traceable elements (of the kinds involved in the respective trace-links defined in Figure 2) can be involved in the associated specific kinds of trace-links, uniqueness (property of UML association classes) is present in the defined trace-links. Due to the binary relations (arity of value 2) in the defined trace-links between the traceable elements, only two elements can be involved in any trace-link. Furthermore, multiplicity of all the traceable elements involved in the trace-links defined is of type "many", since an element can participate in multiple associations (given they are defined by the meta-model and unique).

The main capabilities needed are represented through a feature diagram [9] shown by Figure 3. Storage of trace-links may be internal or external, relative to the prediction models. A traceable element may be of type prediction model element (see Figure 1) or non-model element. Reporting and searching functionality has to be supported. Trace-link info has to include link direction, link meta-data (e.g. date, creator, strength) and cardinality (note that all links are binary, but a single element can be origin or target for more than one trace-link). Typing at the origin and the target ends of a trace-link as well as documenting rationale for trace-link, are optional.

B. Prototype traceability tool

We have developed a prototype tool in the form of a database application with user interfaces, on the top of MS Access [16]. The prototype tool includes a structure of tables for organizing the trace information, queries for retrieval of the trace info, a menu for managing work flow, forms for populating trace-link information, and facilities for reporting trace-links. A screen shot of the entity-relationship (ER) diagram of the trace-link database is shown by Figure 4. The ER diagram is normalized, which means that the data are organized with minimal needs for repeating the entries in the tables. Consistency checks are performed on the referenced fields. The data structure itself (represented by the ER diagram) does not cover all the constraints imposed by the meta-model (shown by Figure 2). However, constraints on queries and forms as well as macros can be added in order to fully implement the logic, such as for example which element types can be related to which trace-link types.

The five traceable element types defined by Figure 2 and their properties (name of creator, date, assumption and comment), are listed in Table *TraceableElementType*. Similarly, the six trace-link types defined by Figure 2 and their properties (scope, date, creator and comment), are listed in Table *TraceLinkType*. Table *TraceableElement* specifies the concrete instances of the traceable elements, and assigns properties (such as the pre-defined element type, hyperlink, creator, date, etc.) to each one of them. Since primary key attribute in Table *TraceableElementType* is foreign key in Table *TraceableElement*, multiplicity between the two respective tables is one-to-many.

Most of the properties are optional, and deduced based on: 1) the core questions to be answered by traceability scheme [9] and 2) the traceability needs for using guidelines for application of prediction models (specified in [4]). The three Tables *TargetElements*, *OriginElements* and *TraceLink* together specify the concrete instances of trace-links. Each link is binary, and directed from a concrete pre-defined traceable element – the origin element specified in Table *OriginElements*, to a concrete pre-defined traceable element – the target element specified in Table *TargetElements*. The trace-link itself (between the origin and the target element) and its properties (such as pre-defined trace-link type) are specified in Table *TraceLink*. Attribute *TraceLinkName* (associated with a unique *TraceLinkId* value) connects the three tables *TraceLink*, *OriginElements* and *TargetElements* when representing a single trace-link instance, thus forming a cross-product when relating the three tables. The MS Access environment performs reference checks on the cross products, as well as on the values of the foreign key attributes. Target elements and origin elements participating in a trace-link, are instances of traceable elements defined in Table *TraceableElement*. They are connected through the Attribute *ElementId* (displayed as *ElementName* in the tables where it has the role of foreign key). Thus, multiplicity between Table *TraceableElement* and Table *TargetElements*, as well as between Table *TraceableElement* and Table *OriginElements*, is one-to-many. Similarly, since primary key attribute in Table *TraceLinkType* is foreign key in Table *TraceLink*, multiplicity between the two respective tables is one-to-many.

A screen shot of the start menu is shown by Figure 5. The sequence of the buttons represents a typical sequence of actions of an end-user (the analyst), in the context of defining, documenting and using the trace-links. The basic definition of the types of the traceable elements and the trace-links are provided first. Then, concrete traceable elements are documented, before defining specific instances of the trace-links and their associated specific origin and target elements, involved in the binary trace-link relations. Finally, reports can be obtained, based on search parameters such as for example model types, model elements, or trace-link types.

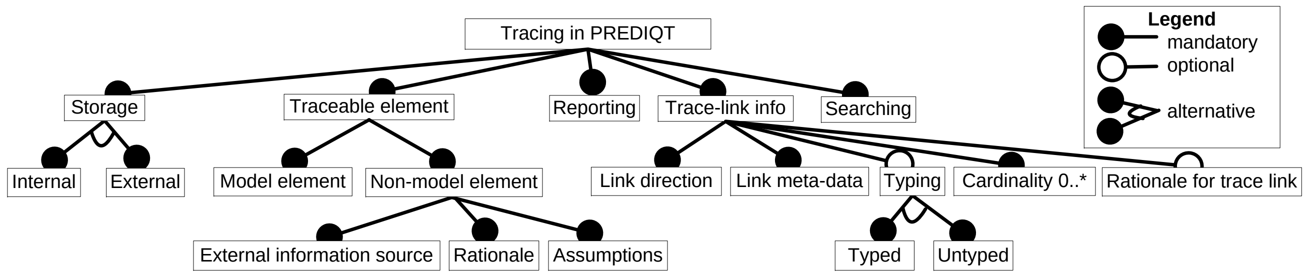


Figure 3. Main capabilities of the traceability approach

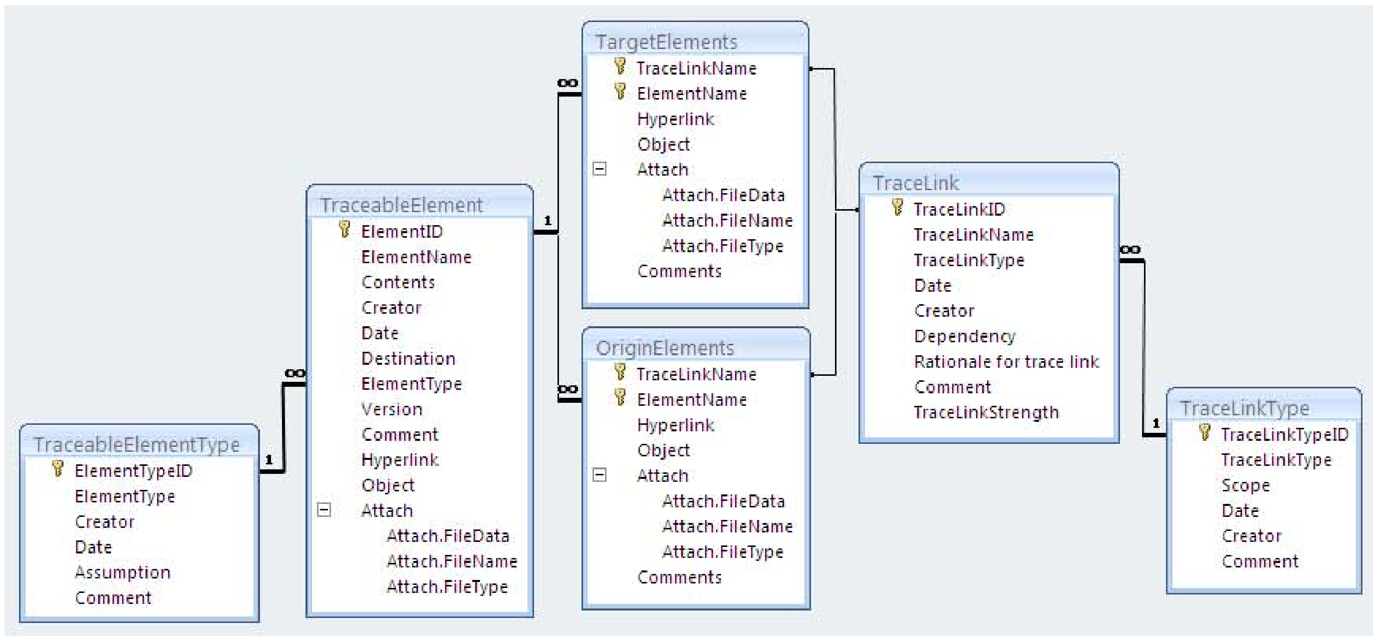


Figure 4. Entity-relationship diagram of the trace-link database of the prototype traceability tool

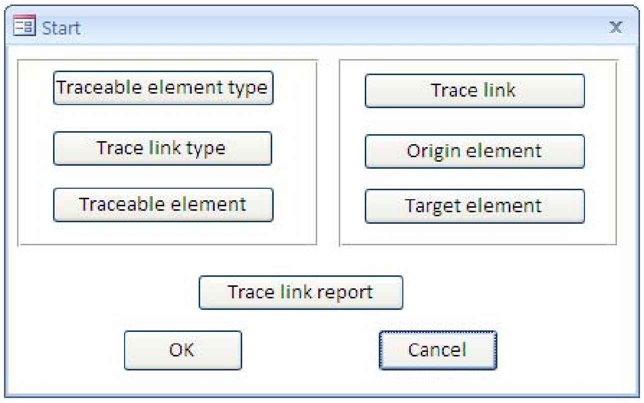


Figure 5. A screen shot of the start menu of the prototype traceability tool

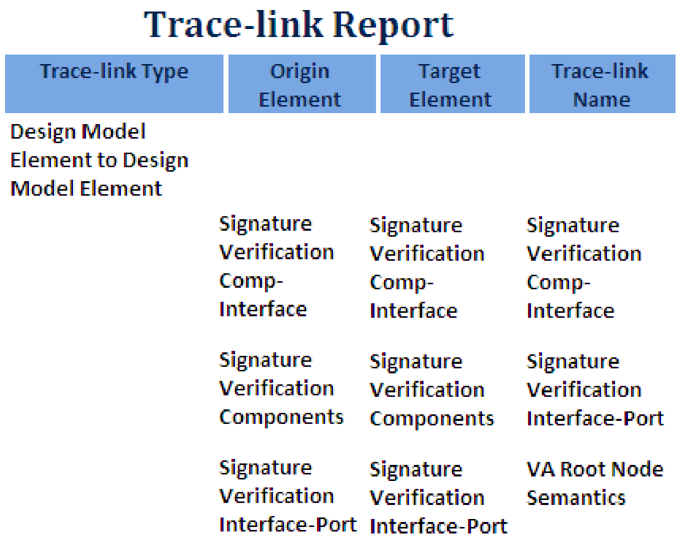
V. Applying the solution on an example

This section exemplifies the application of our solution for managing traces in the context of prediction models earlier developed and applied during a PREDIQT-based analysis [3] conducted on a real-life system.

The trace-link information was documented in the prototype tool, in relation to the model development. The trace-links were applied during change application, according to the guidelines for application of prediction models (specified in [4]). We present the experiences obtained, while the process of documentation of the trace-links is beyond the scope of this paper.

The prediction models involved are the ones related to "Split signature verification component into two redundant components, with load balancing", corresponding to Change 1 in [3]. Three Design Model diagrams were affected, and one, two and one model element on each, respectively. We have tried out the prototype traceability tool on the Design Model diagrams involved, as well as Availability (which was one of the three quality characteristics analyzed) related Quality Model diagrams and DV. Documentation of the trace-links involved within the Availability quality characteristic (as defined by the Quality Model) scope, took approximately three hours. Most of the time was spent on actually typing the names of the traceable elements and the trace-links.

18 instances of traceable elements were registered in the database during the trial: seven Quality Model elements, four DV elements, four Design Model elements and three elements of type "Rationale and Assumptions". 12 trace-links were recorded: three trace-links of type "Design Model Element to Design Model Element", three trace-links of type "Design Model Element to DV Element", one trace-link of type "Design Model Element to Rationale and Assumptions", three trace-links of type "DV Element to Quality Model Element", and two trace-links of type "Structure, Parameter or Semantics of DV Element Documented through Rationale and Assumptions", were documented.



Trace-link Report

Trace-link Type	Origin Element	Target Element	Trace-link Name
Design Model Element to Design Model Element			
	Signature Verification Comp-Interface	Signature Verification Comp-Interface	Signature Verification Comp-Interface
	Signature Verification Components	Signature Verification Components	Signature Verification Interface-Port
	Signature Verification Interface-Port	Signature Verification Interface-Port	VA Root Node Semantics

Figure 6. A screen shot of an extract of a trace-link report from the prototype traceability tool

An extract of a screen shot of a trace-link report (obtained from the prototype tool) is shown by Figure 6. The report included: three out of three needed (i.e., actually existing, regardless if they are recorded in the trace-link database) “Design Model Element to Design Model Element” links, three out of four needed “Design Model Element to DV Element” links, one out of one needed “Design Model Element to Rationale and Assumptions” link, three out of six needed “DV Element to Quality Model Element” links and one out of one needed “Structure, Parameter or Semantics of DV Element Documented through Rationale and Assumptions” link.

Best effort was made to document the appropriate trace-links without taking into consideration any knowledge of exactly which of them would be used when applying the change. The use of the trace-links along with the application of change on the prediction models took totally 20 minutes and resulted in the same predictions (change propagation paths and values of QCF estimates on the Availability DV), as in the original case study [3]. Without the guidelines and the trace-link report, the change application would have taken approximately double time for the same user.

All documented trace-links were relevant and used during the application of the change, and about 73% of the relevant trace-links could be retrieved from the prototype tool. Considering however the importance and the role of the retrievable trace-links, the percentage should increase considerably.

Although hyperlinks are included as meta-data in the user interface for element registration, an improved solution should include interfaces for automatic import of the element names from the prediction models, as well as user interfaces for easy (graphical) trace-link generations between the existing elements. This would also aid verification of the element names.

VI. Why our solution is a good one

This section argues that the approach presented above fulfills the success criteria specified in Section III.

A. *Success Criterion 1*

The traceability scheme and the prototype tool capture the kinds of trace-links and traceable elements, specified in the Success Criterion 1. The types of trace-links and traceable elements as well as their properties, are specified in dedicated tables in the database of the prototype tool. This allows constraining the types of the trace-links and the types of the traceable elements to only the ones defined, or extending their number or definitions, if needed. The trace-links in the prototype tool are binary and unidirectional, as required by the traceability scheme. Macros and constraints can be added in the tool, to implement any additional logic regarding trace-links, traceable elements, or their respective type definitions and relations. The data properties (e.g. date, hyperlink or creator) required by the user interface, allow full traceability of the data registered in the database of the prototype tool.

B. *Success Criterion 2*

Searching based on user input, selectable values from a list of pre-defined parameters, or comparison of one or more database fields, are relatively simple and fully supported based on queries in MS Access. Customized reports can be produced with results of any query and show any information registered in the database. The report, an extract of which is presented in Section V, is based on a query of all documented trace-links and the related elements.

C. *Success Criterion 3*

The text-based fields for documenting the concrete instances of the traceable elements and the trace-links, allow level of detail selectable by the user. Only a subset of fields is mandatory for providing the necessary trace-link data. The optional fields in the tables can be used for providing additional information such as for example rationale, comments, links to external information sources, attachments, strength or dependency. There are no restrictions as to what can be considered as a traceable element, as long at it belongs to one of the element types defined by Figure 2. Similarly, there are no restrictions as to what can be considered as a trace-link, as long at it belongs to one of the trace-link types defined by Figure 2. The amount of information provided regarding the naming and the meta-data, are selectable by the user.

D. *Success Criterion 4*

Given the realism of the prediction models involved in the example, the size and complexity of the target system they address, the representativeness of the change applied on them, the simplicity of the prototype tool with respect to both the user interfaces and the notions involved, as well as the time spent on documenting the trace-links and using them, the application of the approach presented in Section V indicates the applicability of our solution on real-life applications of PREDIQT, with limited resources and by an average user (in the role of the analyst).

The predictions (change propagation paths and values of QCF estimates) we obtained during the application of our solution on the example were same as the ones from the original case study [3] (performed in year 2008) which the models stem from. Although the same analyst has been involved in both, the results suggest that other users should, by following PREDIQT guidelines and applying the prototype traceability tool, obtain similar results.

The time spent is to some degree individual and depends on the understanding of the target system, the models and the PREDIQT method. It is unknown if the predictions would have been the same (as in the original case study) for another user. We do however consider the models and the change applied during the application of the solution, to be representative due to their origins from a major real-life system. Still, practical applicability of our solution will be subject to future empirical evaluations.

VII. Why other approaches are not better in this context

This section evaluates the feasibility of other traceability approaches in the PREDIQT context. Based on our literature review and the results of the evaluation by Galvao and Goknil [10], we argue why the alternative traceability approaches do not perform sufficiently on one or more of the success criteria specified in Section III.

Almeida et al. [17] propose an approach aimed at simplifying the management of relationships between requirements and various design artifacts. A framework which serves as a basis for tracing requirements, assessing the quality of model transformation specifications, meta-models, models and realizations, is proposed. They use traceability cross-tables for representing relationships between application requirements and models. Cross-tables are also applied for considering different model granularities and identification of conforming transformation specifications. The approach does not provide sufficient support for intra-model mapping, thus failing on our Success Criterion 1. Moreover, possibility of representing the various types of trace-links and traceable elements is unclear, although different visualizations on a cross-table are suggested. Tool support is not available, which limits applicability of the approach in a practical setting. Searching and reporting facilities are not available. Thus, it fails on our Success Criteria 1, 2 and 4.

Event-based Traceability (EBT) is another requirements-driven traceability approach aimed at automating trace-link generation and maintenance. Cleland-Huang, Chang and Christensen [18] present a study which uses EBT for managing evolutionary change. They link requirements and other traceable elements, such as design models, through publish-subscribe relationships. As outlined by [10], "Instead of establishing direct and tight coupled links between requirements and dependent entities, links are established through an event service. First, all artefacts are registered to the event server by their subscriber manager. The requirements manager uses its event recognition algorithm to handle the updates in the requirements document and to publish these changes as event to the event server. The event server manages some links between the requirement and its dependent artefacts by using some information retrieval algorithms." The notification of events carries structural and semantic information concerning a change context. Scalability in a practical setting is the main issue, due to performance limitation of the EBT server [10]. Moreover, the approach does not provide sufficient support for intra-model mapping. Thus, it fails on our Success Criteria 1 and 4.

Cleland-Huang et al. [19] propose Goal Centric Traceability (GCT) approach for managing the impact of change upon the non-functional requirements of a software system. Softgoal Interdependency Graph (SIG) is used to model non-functional requirements and their dependencies. Additionally, a traceability matrix is constructed to relate SIG elements to classes. The main weakness of the approach is the limited tool support, which requires manual work. This limits both scalability in a practical setting and searching support (thus failing on our Success Criteria 4 and 2, respectively). It is unclear to what degree granularity of the approach would suffice the needs of PREDIQT.

Cleland-Huang and Schmelzer [20] propose another requirements-driven traceability approach that builds on EBT. The approach involves a different process for dynamically tracing non-functional requirements to design patterns. Although more fine grained than EBT, there is no evidence that the method can be applied with success in a practical real-life setting (required through our Success Criterion 4). Searching and reporting facilities (as required through our Success Criterion 2) are not provided.

Many traceability approaches address trace maintenance. Cleland-Huang, Chang and Ge [21] identify the various change events that occur during requirements evolution and describe an algorithm to support their automated recognition through the monitoring of more primitive actions made by a user upon a requirements set. Mäder and Gotel [22] propose an approach to recognize changes to structural UML models that impact existing traceability relations and, based on that knowledge, provide a mix of automated and semi-automated strategies to update the relations. Both approaches focus on trace maintenance, which is as argued in Section III, not among the traceability needs in PREDIQT.

Ramesh and Jarke [23] propose another requirements-driven traceability approach where reference models are used to represent different levels of traceability information and links. The granularity of the representation of traces depends on the expectations of the stakeholders [10]. The reference models can be implemented in distinct ways when managing the traceability information. As reported by [10], "The reference models may be scalable due to their possible use for traceability activities in different complexity levels. Therefore, it is unclear whether this approach lacks scalability with respect to tool support for large-scale projects or not." In PREDIQT context, the reference models are too broad, their focus is on requirements traceability, and tool support is not sufficient with respect to searching and reporting (our Success Criterion 2).

We could however have tried to use parts of the reference models by Ramesh and Jarke [23] and provide tool support based on them. This is done by [24] in the context of product and service families. The authors discuss a knowledge management system, which is based on the traceability framework by Ramesh and Jarke [23]. The system captures the various design decisions associated with service family development. The system also traces commonality and variability in customer requirements to their corresponding design artifacts. The tool support has graphical interfaces for documenting decisions. The trace and design decision capture is illustrated using sample scenarios from a case study. We have however not been able to obtain the tool, in order to try it out in our context.

A modeling approach by Egyed [25] represents traceability information in a graph structure called a footprint graph. Generated traces can relate model elements with other models, test scenarios or classes [10]. Galvao and Goknil [10] report on promising scalability of the approach. It is however unclear to what degree the tool support fulfills our success criterion regarding searching and reporting, since semantic information on trace-links and traceable elements is limited.

Aizenbud-Reshef et al. [26] outline an operational semantics of traceability relationships that capture and represent traceability information by using a set of semantic properties, composed of events, conditions and actions [10]. Galvao and Goknil [10] state: the approach does not provide sufficient support for intra-model mapping; a practical application of the approach is not presented; tool support is not provided; however, it may be scalable since it is associated with the UML. Hence, it fails on our Success Criteria 1 and 2.

Some approaches [27] [28] [29] that use model transformations can be considered as a mechanism to generate trace-links. Tool support with transformation functionalities is in focus, while empirical evidence of comprehensibility of the approaches in a practical setting, is missing. The publications we have retrieved do not report sufficiently on whether these approaches would offer the searching facilities, the granularity of trace information, and practical applicability needed for use in PREDIQT context (that is, by an analyst who is not an expert in the tools provided).

VIII. Conclusion and future work

Our earlier research indicates the feasibility of the PREDIQT method for model-based prediction of impacts of architectural design changes on system quality. The PREDIQT method produces and applies a multi-layer model structure, called prediction models, which represent system design, system quality and the interrelationship between the two.

Based on the success criteria for a traceability approach in the PREDIQT context, we put forward a traceability scheme. Based on this, a prototype tool which can be used to define, document, search for and represent the trace-links needed, is developed. We have argued that our solution offers a useful and practically applicable support for traceability in the PREDIQT context.

Performing an analysis of factors such as cost, risk, and benefit and following the paradigm of value-based software-engineering, would be relevant in order to stress the effort on the important trace-links. As argued by [9], if the value-based paradigm is applied to traceability, cost, benefit, and risk will have to be determined separately for each trace according to if, when, and to what level of detail it will be needed later. This leads to more important artifacts having higher-quality traceability. There is a trade-off between the semantically accurate techniques on the one hand and cost-efficient but less detailed approaches on the other hand. Finding an optimal compromise is still a research challenge. Our solution proposes a feasible approach, while finding the optimal one is subject to further research.

Further empirical evaluation of our solution is also necessary to test its feasibility on different analysts as well as its practical applicability in the various domains which PREDIQT is applied on. Future work should also include standard interfaces and procedures for updating the traceable elements from the prediction models into our prototype traceability tool. As model application phase of PREDIQT dictates which trace-link information is needed and how it should be used, the current PREDIQT guidelines focus on the application of the prediction models. However, since the group of recorders and the group of users of traces may be distinct, structured guidelines for recording the traces during the model development should also be developed as a part of the future work.

Acknowledgment

This work has been conducted as a part of the DIGIT (180052/S10) project funded by the Research Council of Norway, as well as a part of the NESSoS network of excellence funded by the European Commission within the 7th Framework Programme.

References

[1] A. Omerovic, A. Andresen, H. Grindheim, P. Myrseth, A. Refsdal, K. Stølen, and J. Ølnes, "A Feasibility Study

in Model Based Prediction of Impact of Changes on System Quality," in *International Symposium on Engineering Secure Software and Systems*, vol. LNCS 5965. Springer, 2010, pp. 231–240.

[2] A. Omerovic, B. Solhaug, and K. Stølen, "Evaluation of Experiences from Applying the PREDIQT Method in an Industrial Case Study," in *Fifth IEEE International Conference on Secure Software Integration and Reliability Improvement*. IEEE, 2011.

[3] A. Omerovic, A. Andresen, H. Grindheim, P. Myrseth, A. Refsdal, K. Stølen, and J. Ølnes, "A Feasibility Study in Model Based Prediction of Impact of Changes on System Quality," SINTEF, Tech. Rep. A13339, 2010.

[4] A. Omerovic and K. Stølen, "Traceability Handling in Model-based Prediction of System Quality," SINTEF, Tech. Rep. A19348, 2011.

[5] "Standard Glossary of Software Engineering Terminology: IEEE Std.610. 12-1990," 1990.

[6] A. Knethen and B. Paech, "A Survey on Tracing Approaches in Practice and Research," Frauenhofer IESE, Tech. Rep. 095.01/E, 2002.

[7] N. Aizenbud-Reshef, B. T. Nolan, J. Rubin, and Y. Shaham-Gafni, "Model Traceability," *IBM Syst. J.*, vol. 45, no. 3, pp. 515–526, 2006.

[8] J. Simpson and E. Weiner, *Oxford English Dictionary*. Clarendon Press, 1989, vol. 18, 2nd edn.

[9] S. Winkler and J. von Pilgrim, "A survey of Traceability in Requirements Engineering and Model-driven Development," *Software and Systems Modeling*, vol. 9, no. 4, pp. 529–565, 2010.

[10] I. Galvao and A. Goknil, "Survey of Traceability Approaches in Model-Driven Engineering," in *Proceedings of the 11th IEEE International Enterprise Distributed Object Computing Conference*, 2007.

[11] G. Spanoudakis and A. Zisman, "Software Traceability: A Roadmap," in *Handbook of Software Engineering and Knowledge Engineering*. World Scientific Publishing, 2004, pp. 395–428.

[12] R. J. Wieringa, "An Introduction to Requirements Traceability," Faculty of Mathematics and Computer Science, Vrije Universiteit, Tech. Rep. IR-389, 1995.

[13] N. Anquetil, U. Kulesza, R. Mitschke, A. Moreira, J.-C. Royer, A. Rummler, and A. Sousa, "A Model-driven Traceability Framework for Software Product Lines," *Software and Systems Modeling*, 2009.

[14] S. Bohner and R. Arnold, *Software Change Impact Analysis*. IEEE Computer Society Press, 1996.

[15] J. Rumbaugh, I. Jacobson, and G. Booch, *Unified Modeling Language Reference Manual*. Pearson Higher Education, 2004.

[16] "Access Help and How-to," accessed: May 19, 2011. [Online]. Available: http://office.microsoft.com/en-us/access-help/

[17] J. P. Almeida, P. v. Eck, and M.-E. Iacob, "Requirements Traceability and Transformation Conformance in Model-Driven Development," in *Proceedings of the 10th IEEE International Enterprise Distributed Object Computing Conference*, 2006, pp. 355–366.

[18] J. Cleland-Huang, C. K. Chang, and M. Christensen, "Event-Based Traceability for Managing Evolutionary Change," *IEEE Trans. Softw. Eng.*, vol. 29, pp. 796–810, 2003.

[19] J. Cleland-Huang, R. Settimi, O. BenKhadra, E. Berezhanskaya, and S. Christina, "Goal-centric Traceability for Managing Non-functional Requirements," in *Proceedings of the 27th international conference on Software engineering*. ACM, 2005, pp. 362–371.

[20] J. Cleland-Huang and D. Schmelzer, "Dynamically Tracing Non-Functional Requirements through Design Pattern Invariants," in *Proceedings of the 2nd International Workshop on Traceability in Emerging Forms of Software Engineering*. ACM, 2003.

[21] J. Cleland-Huang, C. K. Chang, and Y. Ge, "Supporting Event Based Traceability through High-Level Recognition of Change Events," *Computer Software and Applications Conference, Annual International*, vol. 0, p. 595, 2002.

[22] P. Mäder, O. Gotel, and I. Philippow, "Enabling Automated Traceability Maintenance through the Upkeep of Traceability Relations," in *Proceedings of the 5th European Conference on Model Driven Architecture - Foundations and Applications*. Springer-Verlag, 2009, pp. 174–189.

[23] B. Ramesh and M. Jarke, "Toward Reference Models for Requirements Traceability," *IEEE Transactions on Software Engineering*, vol. 27, no. 1, pp. 58–93, 2001.

[24] K. Mohan and B. Ramesh, "Managing Variability with Traceability in Product and Service Families," *Hawaii International Conference on System Sciences*, vol. 3, 2002.

[25] A. Egyed, "A Scenario-Driven Approach to Trace Dependency Analysis," *IEEE Transactions on Software Engineering*, vol. 29, no. 2, pp. 116–132, 2003.

[26] N. Aizenbud-Reshef, R. F. Paige, J. Rubin, Y. Shaham-Gafni, and D. S. Kolovos, "Operational Semantics for Traceability," in *Proceedings of the ECMDA Traceability Workshop, at European Conference on Model Driven Architecture*, 2005.

[27] F. Jouault, "Loosely Coupled Traceability for ATL," in *In Proceedings of the European Conference on Model Driven Architecture (ECMDA) workshop on traceability*, 2005, pp. 29–37.

[28] D. S. Kolovos, R. F. Paige, and F. Polack, "Merging Models with the Epsilon Merging Language (EML)," in *MoDELS'06*, 2006, pp. 215–229.

[29] J. Falleri, M. Huchard, and C. Nebut, "Towards a Traceability Framework for Model Transformations in Kermeta," in *Proceedings of the ECMDA Traceability Workshop, at European Conference on Model Driven Architecture*, 2006, pp. 31–40.