A Vision for Product Traceability based on Semantics of Artifacts

Darijus Strašunskas
Dept. of Computer and Information Science, Norwegian Univ. of Science and Technology
Sem Sælands vei 7-9, NO-7491 Trondheim, Norway

Abstract

In the face of extensive attention form both the research community and the industry, traceability there still lacks of a supporting methodology that enables traceability throughout the whole lifecycle of a system. In particular, attention need to be given to geographically distributed development efforts where developers are likely to use different representation formats and a variety of tools for the product development.

An approach to methodological support for artifact management and traceability is presented in this paper. Fragments from different development phases (i.e., requirements specification, design, code, test scenarios, and documentation) are linked to concepts from a domain model and further, interlinked through it. A conceptual domain model is constructed from domain specific concepts (nodes) and quantified relationships between them. An initial domain model and the weights for concept relations are based on the experts' experience and knowledge from previous projects. The main contribution of this work is two fold. First, the approach covers the whole product traceability. Second, prediction and assessment of impact are enabled by tracing related fragments through relations of concepts in a domain model.

Keywords: product traceability, distributed collaborative development, semantic enrichment

1. Introduction

Information system development is a highly iterative process, in which developers try to capture the needs and desires of all stakeholders, and transform them into a complete system, consisting of both manual and computerized parts. The product of such development projects undergoes changes because of their iterative nature. Traceability is defined as a property of a system description technique that allows changes in one of the system descriptions – requirements specification, design, code, documentation, or test scenarios – to be traced to the corresponding fragments of the other descriptions [6]. Such correspondence relationships should be maintained throughout the life time of a system in order to manage the artifact.

Traceability and its relevance to systems development have received much attention in the requirements engineering literature (e.g. [5][18]). Especially, the pre-requirements traceability has been studied severely (e.g. [11][12]). However there is a lack of traceability tools to support the full life-cycle, starting from artifact inception to its use. Different representation formats that are used throughout the development process make it complicated to cover the whole life-cycle of an artifact. Given that a single requirement map to multiple architectural and design concerns which are derived from it, it is difficult to maintain the consistency and traceability. Moreover, an architectural component has a number of relations to various requirements. The task becomes even more difficult in the face of a large system that is build to satisfy thousands of requirements.

In a geographically distributed project developers may use different tools to create and modify product fragments, which can be refined iteratively and further processed by colleagues. Afterwards produced fragments are interchanged among members of a project, so that is important for colleagues to interpret an artifact correctly. These are the main challenges for traceability - to interrelate and trace all artifacts in different representation formats that are produced in a distributed manner using different tools and to cover the whole product lifecycle.

An approach to product fragments management and traceability during the distributed collaborative development process is presented in this paper. Artifacts mapping to the corresponding concepts from a specific problem domain increase the semantics of an artifact. Having the interrelated concepts from the problem domain and all fragments linked to them, it is possible to predict and assess fragment change impact on other fragments.

The overall structure of the paper is as follows. In section two, related works are analyzed. In section three, proposed approach for product traceability is presented. In section four, possible implementations of proposed methodological approach are listed. Finally, in section five, the work is concluded and possible shortcomings and insights how to solve them are discussed.

2. Related Work

Traceability issues have been tackled and a number of techniques have been proposed for providing traceability, such as [5]: cross referencing schemes, based on some form of tagging, numbering, or indexing; requirements traceability matrices. Studies done in the field of traceability have mainly focused on specific parts of the
development process [16] – mostly in the areas of pre-requirements traceability (e.g. [11] [12]) and linking requirements to architectural components (e.g. [7], [13]).

There are approaches based on specific modeling language and/or tool. A much cited tool is TOOR (Traceability of Object-Oriented Requirements), presented by Pinheiro and Goguen in [11], it is based on FOOPS, a formal object-oriented language. Letelier [9] presents a framework for configuring requirements traceability by integrating textual specifications and UML (Unified Modeling Language) model elements. Proposed approach is restricted to UML language and can be applied to software process based on UML.

Some approaches deal with establishing traceability links thereafter the most of a system is developed (e.g. requirements specification, code) and, per se, contribute mainly for product maintenance. Frezza et al [4] propose a system of simulation where both the requirements and implemented system are simulated in order to obtain a set of result data. The data from the requirements and implementation are then compared, which result in a quantitative measure of how accurate the running system implements the requirements. Egyed [2] suggests using a scenario driven approach to acquire runtime information about a system and relate the information – footprints - to the requirements and model of the running system. The footprints are then analyzed in a tool Trace Analyzer, which shows how the components of the system interact when performing specified scenarios. Thus, it is possible to obtain added trace information on how the running system actually fulfills its requirements and which parts of the design are affected.

Ramesh and Jarke in [14] offer a wide vision about the information needed in requirements traceability. Their study is based on the analysis of industrial software development projects. They identify two segments of traceability users and suggest two corresponding traceability meta-models (one is a simplification of the other). In this work the only suggested mechanism to configure the meta-model according to the project needs is to cut or to add parts of the meta-model.

Hence, in general, there is a lack of support and coverage of whole product lifecycle. There is also noticeable disregard of support for distributed teams using different tools and representation techniques and notations. Of course, there are development environments (e.g. Rationl Suite AnalystStudio [15]), which compound together programs for requirements engineering, design, change management and code repository. Though most of integrated programs do not support collaborative work; not all project phases are equally well supported and, by choosing this kind of tool environment, customer is bound to one vendor.

3. Proposed approach - Mapping to the Domain Concept

The objective of this approach is to enable change notification and impact prediction through all phases of development in the distributed projects. That means that different tools and, most likely, different notations will be used during the project. In [3] the list of requirements for product development environments to enable collaboration in geographically distributed developments is emphasized; some of them (relevant to this research) are listed below:

- Unrestricted product object types – a product development environment should allow the developers to share any type of object that they might find useful for supporting their cooperation.
- Unrestricted relation types – a product development environment should allow the developers to create any type of relation between any two objects of product.
- Incremental product refinement – a product development environment should provide the developers with flexible mechanisms for incrementally refining the product. The developers should be allowed to start with vague products, and to refine them into more complete and formal ones.

The traceability approach is based on the requirements to support for collaboration in distributed projects as listed above. There are two basic assumptions underlying as follows:

- CASE-tools (Computer Aided Software Engineering) that are used during the product development support XML (eXtensible Markup Language) or XML-dialect format output of developed fragments. The assumption is reasonable, since most CASE-tools maintain model interchange formats derived from XML.
- There is a problem domain and it can be characterized by well-defined, interrelated concepts. Furthermore these concepts are represented as entities having weighted relationships which show the strength of relationship between the concepts. This assumption is more restrictive since not all entities/relationships can be assigned weights.

Domain model is constructed and all concepts are connected with weighted links according how strongly concepts relate. Those weights further are used to evaluate interrelations between fragments mapped to the domain concepts and to estimate likelihood of impact of one fragment to another. Traceability relations are based on the semantics of the artifacts. Fragments are linked to the concepts from the domain model; all fragments are mapped and linked through the conceptual domain model as follows. There exists a domain model such that:
If fragment $F_i$ is linked to a concept $C_j$ and fragment $F_j$ is linked to $C_k$, then transitively $F_i$ also relates to $F_j$:

$$F_i \rightarrow C_j \land C_j \rightarrow F_j \Rightarrow F_i \rightarrow F_j$$

Having related concepts $C_d$ and $C_b$, and if fragment $F_i$ is linked to a concept $C_d$ and a fragment $F_j$ is linked to $C_b$, then trace dependency in some degree exists between $F_i$ and $F_j$:

$$C_d \rightarrow C_b \land F_i \rightarrow C_d \land C_b \rightarrow F_j \Rightarrow F_i \rightarrow F_j$$

Meta-model for the proposed approach, based on settings described above is depicted in figure 3 using RML (Referent Model Language) [17]. A product is final result of the development project, and it consists of the interrelated products of phase. Product of phase is used to relate specific phase of lifecycle to artifacts developed within the phase (e.g., business analysis, requirements engineering, design, implementation, testing and etc.). Phase products are related by has_change_impact_to relation in order to restrict change notification and propagation only to adjacent phase products. Consider that, developers are notified only about possible impact on fragments from the “surrounding phases”. For example, if a piece of code has been changed, the developers first need to check whether there is some impact on design. Further if design fragment is impacted, then trace back to requirements. Finally, if no impact is present, - change notification because of that change of code stops. It should be noted that relations are not based on sequence of phases in a lifecycle (product development lifecycle is not assumed to be waterfall-like), because a lifecycle usually is highly iterative where phases could be repeated and concurrent where several phases could be developed at the same time. Phase products are related to each other according to the logical dependence between the content of the phase products (e.g., requirements specification and test scenarios).

Stakeholders are responsible of creating and modifying the fragments. A fragment is a semantic piece of phase product in a certain granularity level, e.g., it can be a document, a model, a diagram, a section in a document, a text specifying a non-functional requirement, an use case, a class, an attribute, etc. Links from one fragment to another denote direct dependence between fragments and should be established when possible. Every fragment has semantics, which relate the fragment to one or more domain concept cluster. Weighted mapping relationships are used to distinguish fragment coherency to a particular concept. Domain model is composed from domain concepts. Domain cluster can consist of one or more concepts. Domain concepts are connected by as direct acyclic graphs with weights (weighted relationships). Weights of those relations are calculated based on degree of the concept relatedness.

Figure 1. Product traceability meta-model
The basic steps in approach are (see fig.2):

1. **Building a conceptual domain specific model.** This step consists of two main sub-steps: (a) extraction of domain specific concept and (b) weighing of relationships between concepts (see fig.3).
   - **a)** Syntactical analysis of textual documents has been investigated severely in last few decades. Natural language processing is a main technique used to extract more structural information out of documents. Efforts are directed to build models from requirements specification in natural language. The naïve approach is to use nouns as candidates for entities and verbs for relations between entities. However, there is necessity for more sophisticated techniques to handle linguistic variation when proposing model elements when constructing domain models from a large set of documents. [1] proposes approach of natural language analysis for semantic documents modeling, where techniques for domain model construction are discussed.
   - **b)** Quantification of the relationship between concepts could be done using linguistics and natural language processing techniques for analyzing the documents from a domain. Collocation technique and text mining are used to evaluate the strength of relationship between concepts. The values should be refined by the domain expert – this reflects domain expert’s belief in how much concepts are related in a particular domain. So, these numbers come from either objective data or the experiences of the domain expert accumulated from the development of similar projects. These ranges can be used to represent the high (0.7 to 1.0), medium (0.4 to 0.6) and low (0.0 to 0.3) degree of relation.

2. **Fragmentation of artifacts into semantic fragments.** Produced artifact is translated to XML format and logically fragmented according its semantics. Fragmentation is done by a traceability module which gets the XML file as input and developer defines fragment boundaries. As an output, XML file with added tags to identify start and end positions of fragment is produced.

3. **Fragments mapping to the concepts.** Candidate concepts form domain model are suggested automatically by processing the fragments. Techniques from first step are adapted to extract concepts, if possible, from the fragments and propose the closest related concept from domain model to map to it. Fragments can be linked directly to other fragments if developer finds them related or one fragment is part of another (recall fig.1). Also fragment can be manually linked to domain concept. The weighing scheme is used the same as described in step 1b). Relation information is encoded by XML tags. Finally fragments are stored in a central repository.
4. Application of the approach

Good candidates for implementing the approach for product traceability are Bayesian Belief Network (BBN, also called Bayesian Network or Probabilistic Networks) or weighted graphs. BBN is a powerful technique for reasoning under uncertainty [8][10] and representing knowledge. It provides a graphical model that resembles human reasoning. In last decades Bayesian Belief Network has attracted much attention from both research and industry communities. BBN provides a natural way to structure information about a domain, resembling human reasoning. One advantage of the BBN is that it can not only capture the qualitative relationships among variables (denoted by nodes) but also quantify the conceptual relationships. This is done by assigning conditional probability to each node in the BBN.

Weighted graphs could be used to represent a conceptual domain model. Interrelation of the concepts could be depicted as a distance between concepts. The shortest path algorithm could be used to predict which fragments could be impacted.

5. Discussion and Conclusions

Proposed approach (a) enables full lifecycle product traceability. As nature of collaborative development is usually very iterative, the approach (b) allows tracing and interchanging product fragments at different stages of its incremental refinement (e.g., from abstract sketches to formal representation), (c) does not bind developers to a specific tool and/or modeling language, as far as used tool supports XML output. The use of XML makes it possible to use this approach in settings where the involved artifacts are created and managed by heterogeneous tools, such as text processors and CASE-tools.

Proposal can be beneficial for companies working in the specific domains – a domain model is stable and commonly agreed, expert’s knowledge is available. In case of entering new domain the company should work out specific domain model, which needs to be comprehensible and accepted by all developers. An evolvable domain model is a challenge which should be resolved in future works. Adding or removing some concepts from a conceptual domain model in the middle of project will raise the question what to do with the fragments which are already mapped to that concept. If a new concept is added the similarity between concept and closest fragments could be automatically calculated and the most related fragments re-mapped. Deletion should not remove the concept from domain model, but lock it not allowing mapping new fragments. This would preserve existing links between the concepts and fragments.

Change impact assessment is vital for the large development projects and perhaps the most risky and error-prone task. This approach enables to calculate the probability – how likely some product fragments will be impacted by the change of ‘related’ fragment. That value is calculated based on the weighted relations between domain concepts.

Huge domain model with thousands of concepts could be real challenge for developers to find relevant concept and to link a fragment in question. This issue can be solved by concepts clustering which could ease the finding the right concept. Development of currently hot research area in ontology mapping could also provide useful methods and techniques which could be used both to find the most relevant concept for the fragment and to develop stable and common agreed domain specific model when a domain is new for the developers and several interpretations of domain model exist.

6. Acknowledgment

Special thanks go to my colleague Sari Hakkarainen, who read and corrected the paper and helped to refine the ideas.

7. References


