Software Traceability for Model Driven Architecture

Waraporn Jirapanthong
Faculty of Information Technology
Dhurakij Pundit University
110/1 -4 Prachachuen Road,
Laksi, Bangkok 10210, Thailand
waraporn@it.dpu.ac.th
+66(0)9547300 ext. 207

Abstract

The large number and heterogeneity of models generated during the development of software systems may cause difficulties to identify common and variable aspects among applications, and reuse of core assets available under model driven architecture. In this paper, we advocate the use of traceability relations to support model driven architecture. We propose a rule-based approach to allow for automatic generation of traceability relations with well-defined semantics between different-typed models. In the approach, the rules are represented and the models of concern are represented in XML. Our approach supports various types of traceability relations. We present a traceability reference model with the different types of traceability relations and documents identified in our work, and describe our approach for automatic generation of traceability relations. A prototype tool has been developed to demonstrate and evaluate the approach.

1. Introduction

Model driven architecture (MDA) advocates the use of system models in the various phases of software development process in order to facilitate the creation of families of systems. More specifically, it promotes the separation of system functionality from system implementation on a specific platform. In order to support model driven architecture it is necessary to represent the various models in well defined notations, organise systems in terms of different models, support reuse of best practices to allow for enterprise scale software development, and develop approaches that allows for the (semi-)automatic generation of code from abstract models [4].

As a result, a large number of different models are created when developing software systems following MDA approach. In such setting, it is important to identify and analyse the common and variable aspects of the different models in order to engineer reusable and adaptable components and, therefore, support the development of enterprise-scale applications. However, despite its importance and advances in the area, the support for common and variable aspects among applications and the engineering of reusable and adaptable components are not easy tasks. This is mainly due to the large number and heterogeneity of models generated during the development of systems. Other difficulties are concerned with the (a) necessity of having a basic understanding of the variability consequences during the different development phases of software products by all involved parties, (b) necessity of establishing relationships between models in different levels of abstractions, and (c) poor support for handling complex relations among the models.

The above difficulties are also present when developing software product family applications. In the last years, various methodologies and approaches have been proposed to support the development of software systems based on product family development [10, 12, 16]. These methodologies and approaches are also known as domain-engineering approaches and emphasise a group of related applications in a domain, instead of single applications. More recently, some authors have proposed the use of model driven architectures to assist with product family development [7, 13].

In contrast, requirements traceability has been recognised as an important activity in software system development [11, 27, 28]. In general, traceability relations can improve the quality of the product being developed, and reduce the time and cost associated with the development. In particular, traceability relations can support evolution of software systems, reuse of parts of the system by comparing components of the new and existing systems, validation that a system meets its requirements, understanding of the rationale for certain design and implementation decisions in the system, and analysis of the implications of changes in the system.

However, support for traceability in software engineering environments and tools are not always adequate [28]. Some existing approaches assume manual generation of traceability relations should be established manually [29, 30], which is error-prone, difficult, time consuming, expensive, complex, and limited on expressiveness. Therefore, despite its importance, traceability is rarely established. In order to alleviate this problem, more recently, other approaches have been proposed to support semi- or fully-automatic

generation of traceability relations [1, 6, 9, 17, 21, 26]. However, in the majority of these approaches, the generated traceability relations do not have well-defined semantic meanings necessary to support the benefits provided by traceability.

In this paper, we propose to use traceability to support model driven architecture and cope with the different types of models that are generated during the development of software systems. More specifically, we advocate the use of traceability relations to assist with the identification of common and variable functionality in different models and, therefore, increase the reuse of core assets that are available and reduce inconsistencies between systems.

In order to allow for automatic establishment of traceability relations we propose a rule-based approach in which traceability relations between different types of models are generated. Particularly, the types of models being applied in this paper support specifying different types of models during software development i.e. requirements, analysis and design models.

In this paper, our work focuses on the issues of automatic generation of traceability relations between models produced during the software development. We propose the types of traceability relations to supporting an approach of MDA and present a traceability reference model with seven different types of traceability relations among three types of models concerned with various methodologies. Other differences to our previous work are the development of new set of traceability rules and enabling of traceability relations to supporting MDA approach.

The remaining of this paper is structured as follows. In section 2, we describe a traceability reference model with the main document models and traceability relation types identified in our work. In section 3 we present an overview of our approach, describe the traceability rules, and illustrate the work through examples. In section 4 we discuss some implementation issues and evaluate the work. In section 5 we describe related work. Finally, in section 6 we summarise our approach and discuss directions for future work. Throughout the paper we illustrate our work with examples based on study, analysis, and discussions of mobile phone domain, and ideas in [23, 24].

2. Traceability Reference Model

As in [16], a large number of models are required when developing a set of product family systems. A feature-based approach is important to support domain analysis and domain design, enhance communication between customers and developers in terms of product features, and assist with the development of the family architecture. On the other hand, an object-oriented approach is necessary to assist with the development of the various members in the family. Moreover, those two approaches are applied due to their simplicity, maturity, practicality, and extensibility characteristics.

Specifically, our work concentrates on document models i.e. feature and class diagrams.

Table 1 presents the set of documents used in our work. We assume that for each family of software systems being developed, there is a single instance of each model type at the architecture level (i.e. feature models), but there may exist various instances of the models in the member level (i.e. use cases and class diagrams).

Table 1. Document models used in our approach

Family Architecture Level	Feature model
Family Member Level	Use Cases Class diagram

2.1 Document Models

As below, we briefly describe the document models used in our approach.

Feature Model: A feature model is a document that describes the common and variable aspects (features) of a family of applications in a domain. A feature model is composed of many features. Each feature has a name, a description of the feature in natural language sentences, and a description of possible issues and decisions that may have been raised during the feature analysis process.

As proposed in [19], the features can be classified into four different types namely, (i) capability, (ii) operating environment, (iii) domain technology, and (iv) implementation technique in the feature model. A feature can also be (i) mandatory, when it must exist among applications in a domain, (ii) optional, when it is not necessary in some applications in a domain, or (iii) alternative, when no more than one feature can be selected for an application. Relationships among features can be of type (i) composed_of, (ii) generalisation/specialisation, and (iii) implemented_by.

Use Cases: We propose to represent functional requirements of the various applications as use-cases. Each use case contains a unique *identifier,* information about the family domain (*System* attribute) and *family member identifier*. A use case has also *title, description, level* within a system, *pre- and post-conditions* that must be satisfied before and after its execution respectively, *primary_actor, secondary_actors, flow_of_events* describing the user actions that trigger the use case, *exceptional_events* describing the events that not always occur in the use case, and

superordinate_use_case and subordinate_use_case.

Class Diagrams: The design of each member in a family is described in UML class diagrams. These diagrams are documented according to the standard in [25].

2.2 Traceability Relations

Based on our study and experience with software traceability, and types of traceability relations proposed in [14, 17, 22, 27, 28] , we have identified different types of traceability relations between the various documents that support MDA approach. Those relations are classified in six different groups, as follows.

Group 1: Relations between models in the architecture level and models in the application level (e.g. feature_model vs. use_case).

Group 2: Relations between models of the same type for different applications (e.g. AP1_class_diagram vs. AP2_class_diagram).

Group 3: Relations between models of different types for the same application (e.g. AP1_use_case vs. AP1_class_diagram).

Group 4: Relations between models of different types for different applications (e.g. AP1_use_case vs. AP2_class_diagram).

Group 5: Relations between models of the same type for the same application (e.g. AP1_use_case_UC1 vs. AP1_use_case_UC2).

Each of these groups can assist software development from different perspectives. For instance, relations in group 1 assist with the identification of reusable components in models; relations in group 2 and group 4 support comparisons between the various applications in an architecture; relations in group 3 assist with better understanding of each application in an architecture; and relations in group 5 allow for the identification of evolution aspects in an application and, therefore, supports the decision of when a new application should be created in an architecture.

Table 2 presents a summary of the reference model being proposed in a tabular format. In the table, each cell contains the different types of traceability relations that may exist between the models described in the row and column of that cell. In the table we do not represent the exact elements that are related in the different models, but represent the types of the models. The direction of the relation is represented from a row *[i]* to a column *[j]*. Thus, a relation type *rel_type* in a cell *[i][j]* signifies that "*[i] is related to [j] though rel_type*" (e.g. "class diagram *satisfies* feature model"). The traceability relations that are bi-directional appear in the two correspondent cells for that relation. A brief description of these relations is given below.

Satisfiability: In this type of relation an element e1 *satisfies* an element e2, if e1 meets the expectation and needs of e2. A satisfies relation may hold between (a) an operation or attribute of a class in a class diagram and the description of a use case or feature in a feature model.

Encompass: In this type of relation an element e1 *encompasses* an element e2, if e1 includes the content of e2. This relation exists between feature models and use cases de when a use case includes a feature in the family of applications.

Dependency: In this type of relation an element e1 *depends on* an element e2, if the existence of e1 *relies on* the existence of e2, or if changes in e2 have to be reflected in e1. A depends relation may hold between (a) the description of a use case and the description of a feature in a feature model; (b) an operation or attribute of a class in a class diagram and the description of a use case or feature in a feature model.

Overlap: In this type of relation an element e1 *overlaps* with an element e2, if e1 and e2 refer to common aspects of a system or its domain. This is a bi-directional relation. As shown in Table 2 overlap relations exist between (a) feature models and class diagrams; (b) use case and class diagrams.

Evolution: In this type of relation and element e1 *evolves to* an element e2, if e1 has been replaced by e2 during the development, maintenance, or evolution of the system. An *evolves* relation occurs between document models of the same type for the same application in a family (group 5). This relation may hold between elements in (a) use cases and (b) class diagrams.

Similar: This type of relation occurs between document models of the same type for different applications in a family (group 2). This relation assists with the identification of common aspects between the various members in a family. It is a bi-directional relation that exists between (a) use cases and (b) class diagrams. A *similar* relation between elements e1 and e2 *depends* on the existence of another relation between e1 and e2. For instance, a use case uc1 is *similar* to a use case uc2, if both uc1 and uc2 hold an *encompass* relation with feature f1.

Different: This type of relation also occurs between document models of the same type for different applications in a family (group 2). This relation assists with the identification of variable aspects between the various members in a family. It is a bi-directional relation that exists between (a) use cases and (b) class diagrams. A *different* relation between an element e1

and e2 depends on the existence of another relation between e1 and e2. For instance, a use case uc1 is *different* from a use case uc2, when there are two subclasses c1 and c2 of the same parent class c, where c1 *encompasses* uc1 and c2 *encompasses* uc2. More specifically, consider use cases related to the display of *text message* (uc1) and *display of graphical message* (uc2) on mobile phones; subclasses *TextScreen* (c1) and *GraphicalScreen* (c2), of parent class *ScreenClass* (c). Assume that *ScreenClass* has operation *DisplayMethod*(), which is inherited by classes c1 and c2. In this case, uc1 and uc2 are of the same general purpose (*display of message*), but with different specific aspects (*text* and *graphical messages*).

Table 2. Traceability Reference Model

	Feature Model	Use Case	Class Diagram
Feature Model			*Overlaps*
Use Case	*Encompasses* *Depends_on*	*Similar* *Different* *Evolves*	*Overlaps*
Class Diagram	*Satisfies* *Depends_on* *Overlaps*	*Satisfies* *Depends_on* *Overlaps*	*Similar* *Different* *Evolves*

3. Enabling Traceability Relations

Overview. In order to support automatic generation of the traceability relations described in Section 2, we propose to use a rule-based approach. Rules assist and automate decision making, allow for standard ways of representing knowledge that can be used to infer data, facilitate the construction of traceability generators for large data sets, and support representation of dependencies between elements in the documents.

Figure 1 presents an overview of the process of our approach. Initially, the models are translated into XML format by using an *XML translator*, based on the XML Schemas proposed for the documents, whenever the tools used to create the documents do not generate them in XML originally. The XML translator is also responsible to annotate the textual sentences in the documents with part-of-speech (POS) assignments. The POS are represented as XML tags and generated as XML-based documents. The XML-based documents are used as input to our *traceability generator* that creates traceability relations based on rules.

The traceability relations are also represented in XML. This is important to preserve the original documents, permit the use of these documents by other application and tools, and allow these relations to be used to support identification of other traceability relations that depend on the existence of previously identified relations using the same traceability generator (e.g. *similar* and *different* relations). In the figure, this is represented by using the XML-based-relationships documents as input to the traceability generator.

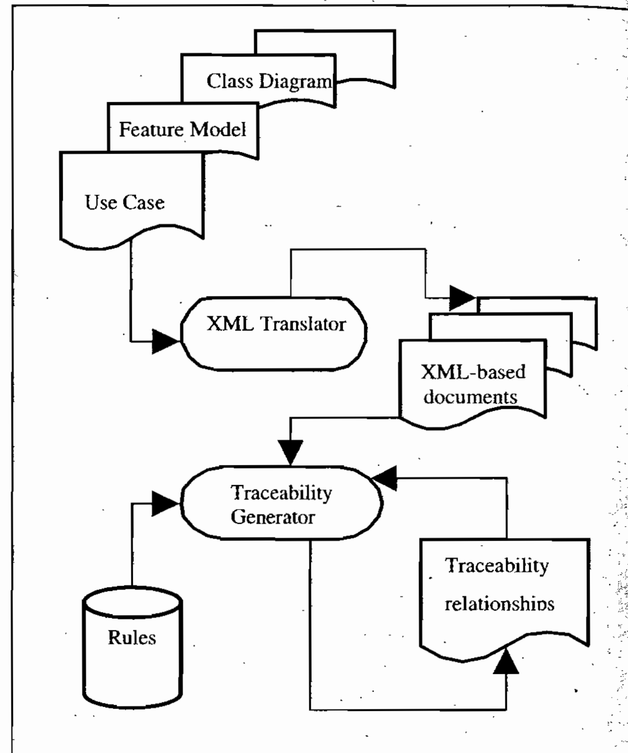

Fig. 1. Overview of the approach

3.1. Traceability Rules

The traceability rules are composed of four main parts, as described below. An example of a traceability rule for *encompass* traceability relation between use cases and features is presented in Figure 2.

Part 1: It consists of the rule identification and contans a unique *RuleID*, description of the type of the rule (*RuleType*), and descriptions of the types of document models associated with the rule (*DocType1, DocType2*). The rule type is based on the type of traceability relation generated by the rule.

Part 2: It consists of declaration statements of variables used in the rule. For the example in Figure 5 we have four variable declarations for *x, y, t1* and *t2*. The variables can be related to each other.

Part 3: It consists of statements which identify elements of the model documents to be compared and bind these elements to variables. At the implementation

level, the elements using XPath expressions associated with placeholders that represent the types of document models. The placeholders for the document models to be traced are automatically substituted by specific model names. The *condition* part of the rule that should be satisfied in order to activate the action part. The condition part uses user-defined functions i.e. *satisfy* function which contains functions e.g. *findSynonym*, returns a list of synonyms for the word passed as a parameter; *checkDistanceControl*, returns "true" or "false" if two or more words are associated in a textual paragraph, depending on how distant the words are in a sentence and what are the other part-of-speech assignments separating these words. The condition part takes into consideration the XML POS-tags in the textual parts of the documents and specifies ways of matching syntactically related terms in the documents.

```
TraceRule  RuleID="R1"
           RuleType="encompass"
           DocType1="Use Case"
           DocType2="Feature Model"
PreCondition
        assign x = Use_Case
        assign y = Feature
        assign t1 = a component in x
        assign t2 = a component in y
Condition
        Satisfy(t1, t2)
Action
        Create
        Relation type = "encompass"
            Element x/Title
            Element y/Feature_name
```

Fig. 2. Example of a traceability rules for *encompass* relation

Part 4: It describes the *action* part of the rule and specifies the action to be taken if the conditions in Part 3 are satisfied. In the consequence part, we describe the type of traceability relation to be created, and the elements that should be related through it in the documents described in the *PreCondition* part of the rule. In the action part, the content of each element is created and together compose with other elements as a relation. The implementation of an action consists of writing the information in the XML-based-relationships document.

An example of the satisfaction of rule R1 exists between use case UC1 and feature. In this case, an *encompass* relation is created. Figure 3 shows the result of rule R1.

```
<Relation_Document>
 <Relation type = "encompass">
   <Element Document="file:///c:/UseCase_UC1.xml">
     <Title> <VVG>Sending</VVG>
         <NN0>Data</NN0></Title>
   </Element>
   <Element Document="file:///c:/Feature_MP.xml">
     <Feature_name><VVG>Messaging</VVG>
             <NN1>Service</NN1></Feature_name>
   </Element>
</Relation>
...
</Relation_Document>
```

Fig. 3. Result of traceability rule R1

In Figure 4 we show an example of a traceability rule for *similar* relations between use cases. This traceability rule depends on the existence of previously identified traceability relations (e.g. *encompass*) between use cases and feature model, described in the XML-based-relationship document. The rule in Figure 4 compares if there are two relations of type *encompass* such that the feature names are the same and the use

```
TraceRule  RuleID="R2"
           RuleType="similar"
           DocType1="XML-Based-Rel"
           DocType2="XML-Based-Rel"
PreCondition
        assign x = XML-based-relationships document
        assign y = XML-based-relationships document
        assign t1 = a traceability relation in x
        assign t2 = a traceability relation in y
        assign c1 = a document model in t1
        assign c2 = a document model in t1
        assign c3 = a document model in t2
        assign c4 = a document model in t2
Condition
     Similar(relationshipType(t1),
             relationshipType(t2),
             c1 ,
             c2,
             c3,
             c4)
Action
        Create
        Relation type = "similar"
               term = relationshipType(t1)
            Element documentModel(c1)
            Element documentModel(c3)
            Element documentModel(c2)
```

Fig. 4. A traceability rule for *similar* relation

case titles are different in these relations. The result of this rule contains the related elements (the names of the use case model) that are identified and generated as a traceability relation.

Suppose that a new mobile phone MP2 is proposed with part of use case UC2. This new product differs from MP1, since it supports General Packet Radio Service (GPRS), but it shares some functionality with MP1.

Based on rule R1, the traceability generator also creates an encompass relation between feature *Messaging Service* and use case UC2. Consider the deployment of rule R2 (Figure 4). As shown in Figure 5, Rule "R2" results in the generation of a *similar* relation between use cases UC1 and UC2, since they both *encompass* feature *Messaging Service*

```
<Relation_Document>
 <Relation type = "similar" term= "encompass" >
   <Element Document="file:///c:/UseCase_UC1.xml">
     <Title> <VVG>Sending</VVG>
          <NN0>Data</NN0></Title>
   </Element>
   <Element Document="file:///c:/UseCase_UC2.xml">
     <Title> <VVG>Sending</VVG>
          <NN0>Text</NN0>
          <NN0>Message</NN0></Title>
   </Element>
   <Element Document="file:///c:/Feature_MP.xml">
    <Feature_name><VVG>Messaging</VVG>
               <NN1>Service</NN1></Feature_name>
   </Element>
</Relation>
...
</Relation_Document>
```

Fig. 5. Result of traceability rule R2

4. Implementation and Evaluation

In order to evaluate and demonstrate our approach we have implemented a prototype tool. We envisage the use of our tool as a general platform for automatic generation of traceability relations and support for model driven architecture. The tool has been implemented in Java and uses Saxon process [31]. The extra functions in our approach have been implemented in Java.

The tool allows the users to select specific models to be traced, but can also establish traceability relations between all the models generated for a family of systems. For each pair of types of selected models, or models developed for a system, the traceability generator component of the tool identifies the traceability rules associated with those models and generate the traceability relations.

The approach provides semantic for the different types of traceability relations that may exist between the models of our concern, and avoids misconception of the relations by the stakeholders. Moreover, the traceability rules used in the tool allows for generation of relations between specific parts of the models (fine granularity), which offers better comparison between the models and identification of the particular assets that can be reused or need to be developed. The traceability rules and relations involve documents in both family architecture and member level. The automatic generation of these relations can provide the deployment of traceability in industrial settings and enhance enterprise-scale system development. The use of the tool facilitates the comparison of large number of complex and heterogeneous document types in a standard and efficient way, supporting scalability of the approach.

Moreover, the tool relies on grammatical structures present in the natural language sentences, which are taken into consideration by the traceability rules. The set of traceability rules should be expanded to allow the generation of relations that consider all possible grammatical structures and, therefore, enhance the recall of the approach. The tool relies on the grammatical structures.

The tool supports the generation of a large number of traceability relations. In order to facilitate the use of the approach, it is necessary to incorporate ways of prioritising the generation and display of the traceability relations, and develop appropriate graphical user interfaces to visualise the relations. Currently, we are developing these graphical user interface applications.

5. Related work

Many approaches and techniques to support software traceability have been proposed. These approaches and techniques can be classified in four main groups: (a) study and definition of different types of traceability relations; (b) support for the generation of traceability relations; (c) development of architectures, tools, and environments for the representation and maintenance of traceability relations; and (d) study of how to use traceability relations to support various software development activities.

Various reference models, frameworks, and classifications have been proposed for different types of traceability relations [11, 14, 17, 20, 22, 27, 28]. The classifications are based on different aspects, ranging

from the types of the related artefacts [9, 15, 17, 27,], to the use of traceability information - in different requirements management activities such as understanding, capture, tracking, evolution, verification, and reuse [6, 11, 22], to impact analysis [34].

However, despite the reference models and classifications there is still a lack of standard semantic definition for the various types of relations. Many existing tools support the representation of the different types of relations, but the interpretation of these relations depends on the stakeholders. The lack of standard semantics causes confusion when interpreting relations and difficulties to develop tool for automatic generation of traceability relations. Moreover, very few classifications for traceability relations in the scope of product family development have been proposed. Some approaches have outlined the use of traceability relations to support product family development [3, 14, 17, 29, 33]. However, these approaches do not provide ways of generating traceability relations automatically. The traceability relation classification proposed in this paper contributes to fulfil the lack of standard semantic. In addition, the tool provides support for interpretation of the automatic generated relations.

The majority of existing tools for the generation of traceability relations offer manual generation of relations based on the use of sophisticated visualisation, display, and navigability components [8, 30]. However, this is error-prone, difficult, time consuming, and expensive, resulting in the rare deployment of traceability relations. Other approaches to support semi- or fully-automatic generation of traceability relations have been proposed [1, 6, 9, 21, 26]. However, none of these approaches support generation of relations for all the types of models supported by our approach.

The approaches to support representation and maintenance of traceability relations range from the use of centralised databases [8, 26, 30] and software repositories [27], open hypermedia architecture [32], mark-up based documents [14], to event-based architecture [6]. As in our previous approach [15], the tool represents the generated traceability relations as XML documents avoiding changes in the original documents and supporting generation of traceability relations that are dependent on existing traceability relations by using the same generator tool.

Software traceability has been used in different stages of the software development life-cycle to assist with various activities. Examples of these activities are change impact analysis, system validation and verification, system reuse, and system understanding [2, 3, 6, 18, 22, 34]. The existing approaches have contributed to the use of traceability relations in various activities in software development. The novelty of our approach is concerned with the automatic generation of different types of traceability relations and the use of these relations to support software development by identifying reusable assets.

6. Conclusion and future work

In this paper we described a rule-based approach to support generation of traceability relations between document models which are created during the software product line applications development. It is believed that the traceability relations can support to reuse the existing models for developing new application in the same domain. The relations between existing document models are identified to present the semantics among models. Those relations assist stakeholders to better understand the semantics of the models. Consequently, the models can be reused and adapted more effectively and efficiently. The idea is driven to support the MDA approach.

We presented a traceability reference model including seven types of traceability relations and three types of documents. We have adopted an extension of XML query language to identify and describe the traceability rules.

Currently, we are extending the set of traceability rules in our approach to identify the proposed traceability relations and enhance recall of the approach. We are identifying and implementing necessary functions, developing a graphical interface to facilitate visualisation of the generated traceability relations, and an editor to support the creation of the traceability relations. We are also investigating ways of prioritising the generation and display of the relations, evaluating the approach in real industrial environments, and extending the work to support traceability for implementation phase of software development.

References

[1] Antoniol G., Canfora G., Casazza G., De Lucia A., Merlo E., "Recovering Traceability Links between Code and Documentation", IEEE Transactions on Software Engineering, 28(10), 970-983, October 2002

[2] Berg, K., and J. Bishop, "Tracing Software Product Line Variability - From Problem to Solution Space", SAICSIT 2005, Pages 111-120.

[3] Biddle R., Noble J., and Tempero E., "Supporting Reusable Use Cases". In Proceedings of the Seventh International Conference on Software Reuse, 2002.

[4] Brown A., "An Introduction to Model Driven Architecture". http://www-106.ibm.com/developerworks/rational/library/3100.html.

[5] Cockburn A., "Structuring Use-Cases With Goals", JOOP, Sep-Oct 1997.

[6] Cleland-Huang J., Chang C.K., Sethi G., Javvaji K., Hu H., Xia J., "Automating Speculative Queries through Event-based Requirements Traceability", proc. of the IEEE Joint International Requirements Engineering Conference, Essen, Germany, September 2002.

[7] Deelstra S., Sinnema M., van Gurp J., Bosh j., "Model Driven Architecture as Approach to Manage Variability in Software Product Families". Proceedings of the Workshop on Model Driven Architecture: Foundations and Applications (MDAFA 2003), pp. 109-114, CTIT Technical Report TR-CTIT-03-27, University of Twente, June 2003.

[8] DOORS. Telelogic DOORS, www.telelogic.com/products/doors.

[9] Egyed A., "A Scenario-Driven Approach to Trace Dependency

Analysis", IEEE Transactions on Software Engineering, Vol.9, No.2, February 2003.

[10] FODA. Feature Oriented Domain Analysis. www.sei.cmu.edu/domain-engineering /FODA.html

[11] Gotel O. and Finkelstein A., "An Analysis of the Requirements Traceability Problem", First International Conference on Requirements, 1994.

[12] Griss M.L., Favaro J., d'Alessandro M., "Integrating Feature Modeling with the RSEB", Proceedings Fifth International Conference on Software Reuse", 1998.

[13] Haugen O., Moller-Pedersen B., Oldevik J., and Solberg A., "An MDA-based Framework for Model-driven Product Derivation". Proceedings of the Eighth IASTED Conference on Software Engineering and Applications (SEA), USA, November 2004.

[14] Jirapanthong W. and A. Zisman, "XTraQue: Traceability for Product Line Systems", Software and Systems Modeling Journal, DOI 10.1007/S10270-007-0066-8, pp 1619-1374 (online, September 2007), pp 1619-1366 (print, to appear).

[15] Jirapanthong, W., "An approach to software artefact specification for supporting product line systems", ISBN: 9789746715741, Dhurakij Pundit University, Bangkok, 2008.

[16] Kang K., "FORM: a feature-oriented reuse method with domain-specific architectures", in Annals of Software Engineering, Vol. 5, pp. 354-355.

[17] Kim, S. D., S. H. Chang, and H. J. La. 2005, "Traceability Map: Foundations to Automate for Product Line Engineering", 3rd ACIS International Conference on Software Engineering Research, Management & Applications(SERA05), Pages 274-281., 2005.

[18] Lavazza L, Valetto G, "Requirements-based Estimation of Change Costs", Empirical Software Engineering - An International Journal, 5(3), November 2000

[19] Lee K., Kang K.C., Chae W., and Choi B.W., "Feature-based Approach to Object-Oriented Engineering of Applications for Reuse", Software-Practice and Experience, 2000, 30:1025-1046.

[20] Lindval M. and Sadahl K., "Practical Implications of Traceability", Software Practice and Experience, Vol. 26, No. 10, pp 1161-1180,1996.

[21] Marcus A., Maletic J.I., "Recovering Documentation-to-Source-Code Traceability Links using Latent Semantic Indexing", ICSE, 2003

[22] Mohan, K. and Ramesh, B., "Managing Variability with Traceability in Product and Service Families", Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS), 2002.

[23] Nokia. http://www.forum.nokia.com/main.html.

[24] OMA. Open Mobile Aliance. www.omg.org/technology/documents/formal/xmi.htm.

[25] OMG. XML Metadata Interchange (XMI). www.omg.org/technology/documents/formal/xmi.htm.

[26] Pinheiro F., Goguen J., "An Object-Oriented Tool for Tracing Requirements", IEEE Software, 52-64, March 1996.

[27] Pohl K., "Process-Centered Requirements Engineering", John Wiley & Sons, Inc., 1996

[28] Ramesh B. and Jarke M., "Towards Reference Models for Requirements Traceability", *IEEE Transactions on Software Engineering* , Vol. 37, No 1. January 2001.

[29] Riebisch M., Plilippow I., "Evolution of Product Lines Using Traceability", OOPSLA 2001 Workshop on Engineering Complex Object-Oriented Systems for Evolution, Florida.

[30] RTM. Integrated Chipware. www.chipware.com.

[31] Saxon. Saxonica. http://saxon.sourceforge.net/.

[32] Sherba S.A., Anderson K.M., and Faisal M., "A Framework for Mapping Traceability Relationships", Proceedings of the 2nd International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE 2003), Canada, September 2003.

[33] Van der Linden, F., "Product Family Development in Philips Medical Systems", Dagstuhl Event 03151, April 2004. www.dagstuhl.de/03151/Titles/index.en.phtml

[34] Von Knethen A., "Automatic Change Support based on a Trace Model", Proceedings of the 1st International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE'02), UK, 2002.