Formalizing Traceability Relations for Product Lines

Luis C. Lamb
Institute of Informatics
Federal University of Rio Grande do Sul
Porto Alegre, 91501-970, Brazil
+55 (51) 3308 9464
luislamb@acm.org

Waraporn Jirapanthong
Faculty of Inf. Technology
Dhurakij Pundit University
Bangkok 10210, Thailand
waraporn@it.dpu.ac.th

Andrea Zisman
School of Informatics
City University London
London EC1V 0HB, UK
+44 2070408346
a.zisman@soi.city.ac.uk

ABSTRACT
Traceability is considered an important activity during the development of software systems. Despite the various classifications that have been proposed for different types of traceability relations, there is still a lack of standard semantic definitions for traceability relations. In this paper, we present an ontology-based formalism for semantic representation of various types of traceability relations for product line systems and associations between these various types of traceability relations.

Categories and Subject Descriptors
D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement – corrections, documentation, enhancement, *reengineering*.

General Terms
Documentation, Design, Languages, Theory.

Keywords
Traceability, relations, product line systems, semantic formalism.

1. INTRODUCTION
Several approaches and techniques have been proposed to support software traceability. These approaches can be classified in four main groups as suggested in [15]: (a) study and definition of different types of traceability relations [5][9][14]; (b) support for generation of traceability relations [2][3][4][6][10][12][16]; (c) development of architectures, tools, and environments for representing and maintaining traceability relations [12][13]; and (d) study of how to use traceability relations in various development activities [3][14]. However, despite its development, there are still many problems and challenges related to different aspects of traceability.

One of these problems is concerned with the need for a formalism to represent the semantics of traceability relations. This problem has been advocated by researchers and practitioners that participated in a series of two workshops [7][8] with the goal of identifying challenges in traceability, and led to the creation of the Grand Challenge document [1]. For semantics of traceability relations, the views of the participants are summarised in [1] as:

TEFSE'11, May 23, 2011, Waikiki, Honolulu, HI, USA

"*C1: In order to effectively utilize links and understand the underlying traceability relationships, it is necessary to define the semantics of a link, however defining a formalism to represent the semantics is a non-trivial task and may be domain specific.*"

The need to capture the semantic of traceability relations is fundamental to provide their effective use. Many existing tools support the representation of different types of relations, but the interpretation of these relations depends on the stakeholders. This causes confusion when interpreting relations and difficulties to develop tools for the automatic generation of relations. Our previous experience with automatic generation of traceability relations [4][10][16] demonstrated that a large number of relations are generated, which may cause difficulties to manage, visualise, and make use of the relations in an effective way. Therefore, it is necessary to provide ways of generating the main traceability relations and inferring other relations based on associations that may exist between these relations.

In previous work [10], we proposed a rule-based framework to allow automatic generation of traceability relations in the scope of product line systems. In [10] we identified nine types of traceability relations for different elements in documents generated when using a feature-based object-oriented engineering approach such as an extension of the FORM (Feature-Oriented Reuse Method) [11] methodology. In this paper, we extend the work in [10], and address the lack of semantic formalism for traceability relations. We propose an ontology-based formalism for different types of traceability relations in the scope of product line systems, and associations among the various types of traceability relations, as discussed in the following sections.

2. FORMALISM AND REASONING
Our work is concerned with a feature-based object-oriented engineering approach to support development of product line systems; i.e., an extension of the FORM methodology [11]. The rationale for using an extension of the FORM methodology is due to its simplicity, maturity, practicality, and extension. A feature-based approach supports domain analysis and design, enhances communication between customers and developers in terms of product features, and assists with the development of product line architectures. An object-oriented approach assists with the development of members in a product line system.

Table 1 summarises the various types of documents used in our work. These documents are classified in two groups, namely (a) documents describing different artefacts at the product line level used by the FORM methodology [11] and (b) documents describing different artefacts at the product member level used by the UML object-oriented notation.

For the documents presented in Table 1, we identified nine types of traceability relations. We classify these traceability relations as *direct* and *indirect* traceability relations. The direct traceability relations are those that do not depend on the existence of other relations and are the *satisfiability*, *dependency*, *overlaps*, *evolution*, *implements*, and *refinement* relations. The indirect traceability relations are those that depend on the existence of other relations and are the *similarity* and *variability* relations.

The domain ontology presented in this paper describes relationships (associations) between documents and reasoning rules one can apply in order to infer traceability relationships. Our inference procedure is based on an initial traceability relational calculus. This procedure identifies association rules between traceability relations

Table 1: Feature-based object-oriented documents

	Domain Analysis	**Domain Design**
Product Line Level	Feature model	Subsystem model
		Process model
		Module model
Product Member Level	Use cases	Class diagram
		Statechart diagram
		Sequence diagram

We present below a formal definition of the traceability relation types. In these definitions an element (e_i) can be either an artefact in a document or a whole document. Due to space restriction we do not describe the specific artefacts that can be associated by each traceability relation type.

SATISFIABILITY: We say that an element e_1 satisfies an element e_2 (denoted by $e_1 \models e_2$) if, and only if e_1 meets the expectations and needs of e_2.

IMPLEMENTS: We say that an element e_1 implements an element e_2 (denoted by $e_1 \vdash e_2$) if e_1 allows for the achievement of e_2.

CONTAINMENT: We say that an element e_1 contains an element e_2 (denoted by $e_1 \supseteq e_2$) if e_1 is a document model that uses an artefact of a design document.

DEPENDENCY: We say that an element e_1 depends on an element e_2 (denoted by $e_1 \partial e_2$) if the existence of e_1 relies on the existence of e_2.

OVERLAPS: We say that an element e_1 overlaps an element e_2 (denoted by $e_1\ \Omega\ e_2$) if e_1 and e_2 refer to common aspects of a system or its domain.

EVOLUTION: We say that an element e_1 evolves to an element e_2 (denoted by $e_1\ \varepsilon\ e_2$) if e_1 has been replaced by e_2 during the development, maintenance or evolution of the system. An evolves to relation exists between two documents of the same type for the same product member.

REFINEMENT: We say that an element e_1 refines an element e_2 (denoted by $e_1\ \rho\ e_2$) if e_1 can be broken down into components and subsystems of e_2; or if e_2 can be specified in more details by e_1.

SIMILARITY: We say that an element e_1 is similar to an element e_2 (denoted by $e_1\ \sigma\ e_2$) if e_1 has a relationship R with an element e3 and e2 also has the same relationship R with element e3, where $R \in \{ \models, \vdash, \supseteq, \partial, \Omega, \varepsilon \}$. The similarity relation occurs between elements of the same types of documents for different product members.

VARIABILITY: We say that an element e_1 is variable from an element e_2 (denoted by $e_1 \mathrel{!=} e_2$) if e_1 has a relationship R with an element e3, e2 also has the same relationship R with element e4, and elements e3 and e4 are variants of the same variability point, where $R \in \{ \models, \vdash, \supseteq, \partial, \Omega, \varepsilon \}$. The variability relation occurs between elements of the same types of documents for different product members.

Based on our definitions of the semantics of the different types of traceability relations, we identified several associations between these relation types, namely (a) *implication,* (b) *weak implication,* and (c) *derivation* associations. These associations correspond to the reasoning method of our ontology. We define an *implication* association as the logical implication relation in which A -> B (*A "implies" B*) meaning that every value of A is also a value of B. We define a *weak implication* association A ---> B (*A "weakly implies" B*) when a sub-set of the values of A are also values of B. We define a *derivation* association A ==> B (*A "derives" B*) for the indirect traceability relations (i.e., similarities and variability relations) that are derived from direct relations.

Figure 1 shows a graph of the *implication* associations for the types of traceability relations. An *implication* association exists between (i) implements and satisfiability relations, (ii) implements and dependency relations, (iii) implements and overlaps relations, (iv) containment and dependency relations, (v) containment and overlaps relations, and (vi) similarity and overlaps relations.

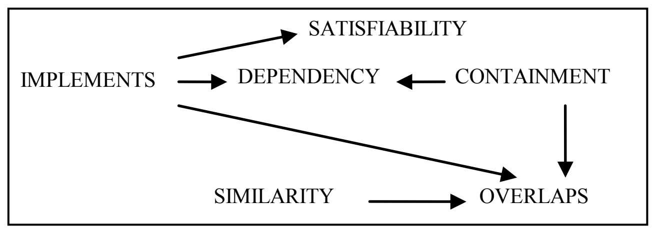

Figure 1: Implication associations

Figure 2 shows a graph of the *weak implication* associations between the traceability relations. A weak implication association exists between (i) refinement and overlaps relations, (ii) refinement and dependency relations, (iii) refinement and containment relations, (iv) refinement and satisfiability relations, (v) overlaps and dependency relations, (vi) overlaps and satisfiability relations, (vii) evolution and refinement relations, and (viii) implements and refinements relations. We describe below the cases in which these weak associations hold.

Refinement and Overlaps: A refinement relation between two elements e_1 and e_2 ($e_1\ \rho\ e_2$) is also an overlaps relation between e1 and e2 ($e_1\ \Omega\ e_2$), when

- e_1 is a sequence diagram and e_2 is a class diagram;
- e_1 is a statechart diagram and e_2 is a sequence diagram;
- e_1 is a statechart diagram and e_2 is a class diagram;
- e_1 is a class diagram and e_2 is a subsystem model;
- e_1 is a sequence diagram and e_2 is a process or module model;
- e_1 is a statechart diagram and e_2 is process or module model.

Refinement and Dependency: A refinement relation between two elements e_1 and e_2 ($e_1\ \rho\ e_2$) is also a dependency relation between e1 and e2 ($e_1 \partial e_2$), when

- e_1 is a statechart diagram and e_2 is a class diagram;
- e_1 is a sequence diagram and e_2 is a class diagram.

Refinement and Containment: A refinement relation between two elements e_1 and e_2 ($e_1\ \rho\ e_2$) is also a containment relation between e1 and e2 ($e_1 \supseteq e_2$), when

- e_1 is a sequence diagram and e_2 is a class diagram;
- e_1 is a statechart diagram and e_2 is a class diagram;
- e_1 is a class diagram and e_2 is a subsystem model.

Refinement and Satisfiability: A refinement relation between two elements e_1 and e_2 ($e_1\ \rho\ e_2$) is also a satisfiability relation between e1 and e2 ($e_1 \models e_2$), when e_1 is a subsystem, process, or module model and e_2 is a feature in a feature model.

Overlaps and Dependency: An overlaps relation between two elements e_1 and e_2 ($e_1\ \Omega\ e_2$) is also a dependency relation between e1 and e2 ($e_1\ \partial\ e_2$), when e_1 is a sequence of events in a sequence diagram or a transition in a statechart diagram, and e_2 is a message in a process model or module model.

Overlaps and Satisfiability: An overlaps relation between two elements e_1 and e_2 ($e_1\ \Omega\ e_2$) is also a satisfiability relation between e1 and e2 ($e_1 \models e_2$), when

- e_1 is a class or operation in a class diagram and e_2 is a feature in a feature model;
- e_1 is a transition in a statechart diagram and e_2 is a feature in a feature model;
- e_1 is a sequence of messages in a sequence diagram and e_2 is a feature in a feature model;
- e_1 is a class or operation in a class diagram and e_2 is a use case;
- e_1 is a transition in a statechart diagram and e_2 is a use case;
- e_1 is a sequence of messages in a sequence diagram and e_2 is a use case;
- e_1 is a subsystem, process, or module model and e_2 is a feature in a feature model.

Evolution and Refinement: An evolution relation between two elements e_1 and e_2 ($e_1\ \varepsilon\ e_2$) is also a refinement between e1 and e2 ($e_1\ \rho\ e_2$), when e_1 is considered a specialization of e_2.

Implements and Refinement: An implements relation between two elements e_1 and e_2 ($e_1 \vdash e_2$) is also a refinement relation between e1 and e_2 ($e_1\ \rho\ e_2$), when e_1 provides more details on how e_2 can be executed.

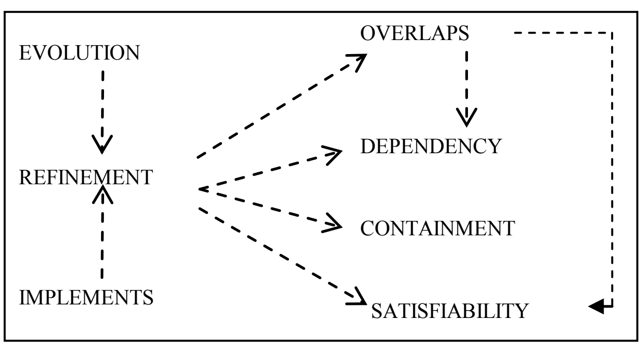

Figure 2: Weak implication associations

Figure 3 shows a graph of the *derivation* associations of the similarity relations. The similarity relations can be derived from other relations by inference rules or conjunctions of relationships. The inference rules that provide the derivation are:

- Satisfiability *derives* Similarity. If element e_1 satisfies element e_2, and element e_3 satisfies element e_2, then e_1 is similar to e_3.
- Implements *derives* Similarity. If element e_1 implements element e_2 and element e_3 implements element e_2, then e_1 is similar to e_3.
- Containment *derives* Similarity. If element e_1 contains element e_2 and element e_3 contains element e_2, then e_1 is similar to e_3.
- Overlaps *derives* Similarity. If element e_1 overlaps element e_2 and element e_3 overlaps element e_2, then e_1 is similar to e_3.

The conjunction of the evolution relation with another relation can lead to the derivation of similarity relations, as follows.

- Evolution with Overlaps. If e_1 evolves to e_2, e_3 evolves to e_4, e_2 overlaps with e_5, and e_4 overlaps with e_5, then e_2 is similar to e_4.
- Evolution with Containment. If e_1 evolves to e_2, e_3 evolves to e_4, e_2 contains e_5, and e_4 contains e_5, then e_2 is similar to e_4.
- Evolution with Implements. If e_1 evolves to e_2, e_3 evolves to e_4, e_2 implements e_5, and e_4 implements e_5, then e_2 is similar to e_4.
- Evolution with Satisfiability. If e_1 evolves to e_2, e_3 evolves to e_4, e_2 satisfies e_5, and e_4 satisfies e_5, then e_2 satisfies e_4.

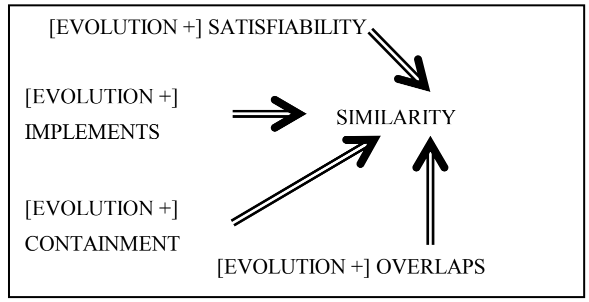

Figure 3: Derivation associations for similarity relations

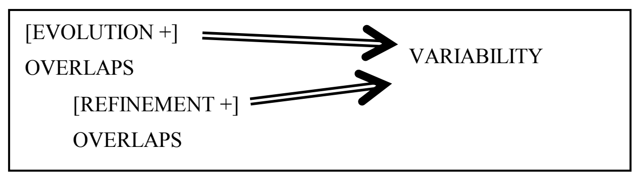

Figure 4: Derivation associations for variability relations

As in the case of the similarity, the variability relations can also be derived from other relations by conjunctions of relationships. The conjunction relations that can lead to the derivation of variability relations are described below and shown in Figure 4.

- Evolution with Overlaps. If e_1 evolves to e_2, e_3 evolves to e_4, e_2 overlaps e_5, e_4 overlaps e_6 and e_5 is a sub-type element of e_6, then e_2 is a variation of e_4.
- Refines with Overlaps. If e_1 refines e_2, e_3 refines e_4, e_2 overlaps e_5, e_4 overlaps e_6, and e_5 is a sub-type element of e_6, then e_2 is a variation of e_4.

3. IMPLEMENTATION & RESULTS

We implemented a prototype tool to allow automatic generation of traceability relations for product line systems. The tool uses traceability rules specified in an extension of XQuery to generate traceability relations. The rules take into consideration the (i) semantics of the documents, (ii) various types of traceability relations, (ii) grammatical roles of the words in the textual parts of the documents, and (iv) synonyms and distances of words being compared in a text. The tool has been implemented in Java and uses SAXON to evaluate XQuery rules.

We used the tool to automatically generate traceability relations between the documents in three different scenarios for product line engineering, namely (a) S1: creation of a new product member for an existing product line, (b) S2: creation of a product line from existing product members, and (c) S3: changes to a product member in a product line. We used a case study of mobile phone product line system with three product members (P1, P2, P3) with common and variable characteristics. Table 2 describes a summary of the types and number of documents and their main elements used in the case study. Table 3 presents, for each different scenario, the total number of traceability relations generated by the tool. An empty cell signifies that the respective traceability relation type was not generated.

Table 2: Number of documents and elements in the case study

Level	Documents	Numbers			Elements	Numbers		
Product Line Models	Feature	1			Features	130		
	Subsystem	1			Subsystems	5		
	Process	6			Processes	48		
	Module	15			Modules	167		
Product Member		P1	P2	P3		P1	P2	P3
	Use Cases	4	4	5	Events	37	36	44
	Class Diagram	1	1	1	Classes	23	25	27
					Attributes	26	26	33
					Methods	78	82	87
	Sequence Diagram	4	4	5	Messages	114	82	112
					Objects	22	21	27
	Statechart Diagram	1	1	1	States	4	4	8
					Transitions	8	8	8

Table 3: Number of traceability relations for S1, S2, and S3

Relations	Scenario S1	Scenario S2	Scenario S3
Implements	172	410	-
Satisfiability	154	322	-
Containment	19	16	20
Refinement	180	342	60
Dependency	-	-	28
Overlaps	-	-	28
Total Direct	525	1090	136
Similarity	333	1402	130
Variability	341	16	-
Total Indirect	674	1418	130
TOTAL	1199	2508	166

The manual analysis of the relations generated by the tool, demonstrated that (a) for scenarios S1 and S2 implements relations imply the satisfiability relations; and (b) for scenario S3 containment relations imply overlaps relations, and similar relations imply overlaps relations. The analysis confirmed implication associations between containment and dependency relations and containment and overlaps relations for scenario S3. Moreover, the analysis demonstrated validity of the weak implications between refinement and containment, refinement and satisfiability, and implication and refinement relations in scenarios S1 and S2; and refinement and overlaps, refinement and containment, and refinement and dependency relations for scenario S3. The derivation of similarity relations due to inference rules, were confirmed for scenarios S1 and S2 for the cases of implements, containment, and satisfiability relations. The derivation of similarity relations was demonstrated for containment and overlaps relations in scenario S3.

4. CONCLUSIONS AND FUTURE WORK

We presented a formalism for traceability relations in the domain of product line systems. We identified and defined nine types of traceability relations and presented three different types of associations that may exist between these traceability relations. Currently, we are extending the work to provide traceability relation types for the domain implementation phase of product line systems and for agent-oriented and knowledge-based systems. We are also expanding our tool to support the types of associations between relations presented in the paper.

5. REFERENCES

[1] Antoniol, G., Berenbach, B., Cleland-Huang, J., Dekhtyar, A., Egyed, A., Fergunson, S., Huffman, J., Maletic, J., and. Zisman, A. 2007. www.traceabilitycenter.org/downloads/ documents/GrandChallenges.

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

[3] Cleland-Huang, J., Chang, C., Sethi, G., Javvaji, K., Hu, H., Xia, J. 2002. Automating Speculative Queries through Event-based Requirements Traceability. IEEE Joint Intl. Requirements Engineering Conference,Germany, September.

[4] Cysneiros, G. and Zisman, A. 2007. Tracing Agent-Oriented Systems. Grand Challenge Traceability Symposium, March.

[5] Dick, J. 2002. Rich Traceability. Workshop on Traceability for Emerging forms of Software Engineering, UK.

[6] Egyed A. 2003. A Scenario-Driven Approach to Trace Dependency Analysis, IEEE Transactions on Software Engineering, Vol.9, No.2, February.

[7] GCT'06. First Workshop on Grand Challenges for Traceability, NASA, Fairmont, West Virginia, USA, 2006.

[8] GCT'07. International Symposium on Grand Challenges for Traceability, Kentucky, USA, 2007.

[9] Gotel, O. and Finkelstein, A. 1994. An Analysis of the Requirements Traceability Problem, International Conference on Requirements Engineering, USA.

[10] Jirapanthong, W. and Zisman, A. 2009. XTraQue: Traceability for Product Line Systems. Software and System Modeling Journal, 8 (1), pp. 1619-1366, February.

[11] Kang, K., Kim, S., et al. FORM: A Feature-Oriented Reuse Method with Domain-Specific Architectures, Annals of Software Engineering 5(1): 143-168.

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

[13] Pohl, K. 1996. Process-Centered Requirements Engineering," Wiley/Research Studies Press, New York.

[14] Ramesh B. and Jarke M. 2001. Towards Reference Models for Requirements Traceability, IEEE Transactions on Software Engineering, vol. 37.

[15] Spanoudakis, G. and Zisman, A. 2005. Software Traceability: A Roadmap, in S. K. Chang, ed., Handbook of Software Engineering and Knowledge Engineering.

[16] Spanoudakis, G., Zisman, A., Pérez-Miñana, E., and Krause, P. 2004. Rule-based Generation of Requirements Traceability Relations, Journal of Systems and Software, 72.