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
warapom@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:

“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

<table>
<thead>
<tr>
<th>Product Line Level</th>
<th>Domain Analysis</th>
<th>Domain Design</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Feature model</td>
<td>Subsystem model</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Process model</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Module model</td>
</tr>
<tr>
<td>Product Member Level</td>
<td>Use cases</td>
<td>Class diagram</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Statechart diagram</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sequence diagram</td>
</tr>
</tbody>
</table>

We present below a formal definition of the traceability relation types. In these definitions an element (e) 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 e₁ satisfies an element e₂ (denoted by e₁ ⊨ e₂) if, and only if, e₁ meets the expectations and needs of e₂.

**Implements:** We say that an element e₁ implements an element e₂ (denoted by e₁ ⊨ e₂) if e₁ allows for the achievement of e₂.

**Containment:** We say that an element e₁ contains an element e₂ (denoted by e₁ ⊇ e₂) if e₁ is a document model that uses an artefact of a design document.

**Dependency:** We say that an element e₁ depends on an element e₂ (denoted by e₁ ⊨ e₂) if the existence of e₁ relies on the existence of e₂.

**Overlaps:** We say that an element e₁ overlaps an element e₂ (denoted by e₁ ∩ e₂) if e₁ and e₂ refer to common aspects of a system or its domain.

**Evolution:** We say that an element e₁ evolves to an element e₂ (denoted by e₁ ∈ e₂) if e₁ has been replaced by e₂ 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 e₁ refines an element e₂ (denoted by e₁ ≺ e₂) if e₁ can be broken down into components and subsystems of e₂, or if e₂ can be specified in more details by e₁.

**Similarity:** We say that an element e₁ is similar to an element e₂ (denoted by e₁ ∼ e₂) if e₁ has a relationship R with an element e₃, and e₂ also has the same relationship R with e₃, where R ∈ { ⊨, ⊇, ≺, ∩, ∼, ⊂, ⊃, ⊆, ⊇, ≺, ∩, ∼ }. The similarity relation occurs between elements of the same types of documents for different product members.

**Variability:** We say that an element e₁ is variable from an element e₂ (denoted by e₁ ≠ e₂) if e₁ 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 ∈ { ⊨, ⊇, ≺, ∩, ≃, ⊂, ⊃, ⊆, ⊇, ≺, ∩, ≃ }. 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 e₁ and e₂ (e₁ ⊨ e₂) is also an overlaps relation between e₁ and e₂ (e₁ ∩ e₂), when
- e₁ is a sequence diagram and e₂ is a class diagram;
- e₁ is a statechart diagram and e₂ is a sequence diagram;
- e₁ is a statechart diagram and e₂ is a class diagram;
- e₁ is a class diagram and e₂ is a subsystem model;
- e₁ is a sequence diagram and e₂ is a process or module model;
- e₁ is a statechart diagram and e₂ is process or module model.

**Refinement and Dependency:** A refinement relation between two elements e₁ and e₂ (e₁ ≺ e₂) is also a dependency relation between e₁ and e₂ (e₁ ≺ e₂), when
- e₁ is a statechart diagram and e₂ is a class diagram;
- e₁ is a sequence diagram and e₂ is a class diagram.

**Refinement and Containment:** A refinement relation between two elements e₁ and e₂ (e₁ ⊂ e₂) is also a containment relation between e₁ and e₂ (e₁ ⊂ e₂), when
• e₁ is a sequence diagram and e₂ is a class diagram;
• e₁ is a statechart diagram and e₂ is a class diagram;
• e₁ is a class diagram and e₂ is a subsystem model.

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

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

Overlaps and Satisfiability: An overlaps relation between two elements e₁ and e₂ (e₁ ⊨ e₂) is also a satisfiability relation between e₁ and e₂ (e₁ ⊨ e₂), when
• e₁ is a class or operation in a class diagram and e₂ is a feature in a feature model;
• e₁ is a transition in a statechart diagram and e₂ is a feature in a feature model;
• e₁ is a sequence of messages in a sequence diagram and e₂ is a feature in a feature model;
• e₁ is a class or operation in a class diagram and e₂ is a use case;
• e₁ is a sequence of messages in a sequence diagram and e₂ is a use case;
• e₁ is a subsystem, process, or module model and e₂ is a feature in a feature model.

Evolution and Refinement: An evolution relation between two elements e₁ and e₂ (e₁ ⊨ e₂) is also a refinement relation between e₁ and e₂ (e₁ ⊨ e₂), when e₁ is considered a specialization of e₂.

Implements and Refinement: An implements relation between two elements e₁ and e₂ (e₁ ⊨ e₂) is also a refinement relation between e₁ and e₂ (e₁ ⊨ e₂), when e₁ provides more details on how e₂ can be executed.

Overlaps derives Similarity. If element e₁ overlaps element e₂ and element e₁ overlaps element e₂, then e₁ is similar to e₂.

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

• Evolution with Overlaps. If e₁ evolves to e₂, e₁ evolves to e₃, e₂ overlaps with e₄, and e₃ overlaps with e₄, then e₁ is similar to e₃.
• Evolution with Containment. If e₁ evolves to e₂, e₂ evolves to e₄, e₃ contains e₄, and e₅ contains e₄, then e₁ is similar to e₂.
• Evolution with Implements. If e₁ evolves to e₂, e₁ evolves to e₃, e₂ implements e₄, and e₃ implements e₅, then e₁ is similar to e₂.
• Evolution with Satisfiability. If e₁ evolves to e₂, e₁ evolves to e₃, e₂ satisfies e₄, and e₃ satisfies e₅, then e₁ is similar to e₂.

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 e₁ evolves to e₂, e₁ evolves to e₃, e₂ overlaps e₅, e₄ overlaps e₆, and e₁ is a sub-type element of e₆, then e₂ is a variation of e₆.
• Refines with Overlaps. If e₁ refines e₂, e₂ refines e₃, e₂ overlaps e₅, e₄ overlaps e₆, and e₁ is a sub-type element of e₆, then e₂ is a variation of e₆.

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, (iii) 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

<table>
<thead>
<tr>
<th>Level</th>
<th>Documents</th>
<th>Numbers</th>
<th>Elements</th>
<th>Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product Line</td>
<td>Feature</td>
<td>1</td>
<td>Features</td>
<td>130</td>
</tr>
<tr>
<td></td>
<td>Subsystem</td>
<td>1</td>
<td>Subsystems</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Process</td>
<td>6</td>
<td>Processes</td>
<td>48</td>
</tr>
<tr>
<td></td>
<td>Module</td>
<td>15</td>
<td>Modules</td>
<td>167</td>
</tr>
<tr>
<td>Use Cases</td>
<td>P1</td>
<td>4</td>
<td>Events</td>
<td>37</td>
</tr>
<tr>
<td></td>
<td>P2</td>
<td>4</td>
<td></td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>P3</td>
<td>5</td>
<td></td>
<td>44</td>
</tr>
<tr>
<td>Class Diagram</td>
<td>Diagram</td>
<td>1</td>
<td>Classes</td>
<td>23</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Attributes</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Methods</td>
<td>78</td>
</tr>
<tr>
<td>Sequence</td>
<td>Diagram</td>
<td>4</td>
<td>Messages</td>
<td>114</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>Objects</td>
<td>82</td>
</tr>
<tr>
<td>Statechart</td>
<td>Diagram</td>
<td>1</td>
<td>States</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Transitions</td>
<td>8</td>
</tr>
</tbody>
</table>

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

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

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