A Model for Conformance Analysis of Software Documents[*]

Tien N. Nguyen and Ethan V. Munson
Department of EECS
University of Wisconsin-Milwaukee
{tien, munson}@cs.uwm.edu

Abstract

During the evolution of a large-scale software project, developers produce a large variety of software artifacts such as requirement specifications, design documents, source code, documentation, bug reports, etc. These software documents are not isolated items — they are semantically related to each other. They evolve over time and the set of active semantic relationships among them is also dynamic. Their evolution makes the task of managing them and maintaining their semantic consistency a very challenging task for software developers. At times, the evolutionary changes may reduce the consistency of the software project and break semantic connections among its documents. We use the term conformance *to denote the state where the network formed by software documents and their relationships is in semantic harmony.* Conformance analysis *is the process of determining whether software documents and their logical relationships are in agreement.*

In this paper, we present a representation for software documents and their logical relationships based on the hypertext model. We describe how conformance analysis can be supported by this representation and present a method to detect non-conformance using timestamps and versioned hypermedia. Then we describe a formalism, called the conformance model, *that can be used to understand and combine approaches to the conformance analysis problem.*

1 Introduction

Developers working on a large-scale software project produce a large variety of software artifacts using different management tools in various environments. Regardless of the process model that is chosen, software documents typically include requirement documents, feasibility studies, business plans, design specifications, source code, module documentation, test plans, bug reports, sales plans, etc. The field of software engineering concerns itself with the task of managing this collection so that information is accessible, correct and consistently organized.

An important factor that makes this task challenging is the dynamic characteristic of a large software project. The software artifacts always evolve over time and the set of active logical relationships among them is also dynamic. One software document may be logically related to another, which in turn is involved in relationships with one or more other documents. For instance, a requirement document A "motivates" a design document B, which "requires" the use of a class C, while a bug report D "complains" about a function E, which is later modified into new revision E' "in response to" the bug report D. Therefore, documents and their relationships can be seen as a network. The state of the network in which projects' software documents are in perfect semantic harmony with each other and with the semantics of their relationships is referred as *conformance*. The evolution of these documents may result in the breaking of the conformance state of the project and of the semantic connections among documents as well. We use the term *conformance analysis* to refer to the process of determining whether software documents and their logical relationships are in agreement.

In current practice, the lack of powerful and automated tools to maintain conformance forces developers to use ad hoc methods, including paper notes or memorization, which can lead to cognitive overhead and mistakes. This problem hinders developers from having a full understanding of the system and from discovering important new information. Therefore, it reduces developers' effectiveness and makes many software maintenance tasks more difficult than they might be if better tools were available.

To explore solutions to the conformance problem, we have been developing a new environment for software documents, the Software Concordance (SC). This system uses a uniform XML-compatible format to represent all types of software documents (including Java source code), uses hyperlinks to represent relationships between documents and

[*]This research was supported by the UWM Dissertation Fellowship, the U. S. Department of Defense, NASA and by NSF CAREER awards CCR-9734102.

maintains a fine-grained version history for document elements and the links between them. This versioned hypermedia [17] infrastructure will support experiments on conformance analysis mechanisms and interfaces.

This paper presents our efforts to develop a formal basis for conformance analysis. The next section discusses how conformance analysis is related to the domains of requirements traceability and inconsistency management and also presents Gunter's model of the software system build process [11]. Section 3 discusses document relationships and how the hypertext model can be used to represent them in a manner suitable for conformance analysis. Section 4 describes our current approach to implementing conformance analysis. In Section 5, we present our formal model for conformance analysis, the *conformance model*, which is based on Gunter's model of the build process. Our conclusions appear in the final section.

2 Related work

2.1 Requirements traceability and inconsistency management

Conformance analysis is related to the well-known processes of requirements traceability [8, 34] and inconsistency management [9, 19, 29].

Requirements traceability and its importance in software development process have been well described [8, 34]. A number of requirements tracing tools have been developed and integrated into software development environments [22, 23]. Other research seeks to develop a reference model for requirements traceability that defines types of requirement documentation entities and traceability relationships [14, 25, 32]. Dick [5] extends the traceability relationships to support more consistency analyses.

Inconsistency management and *impact analysis* have been studied since the late 1980s [29]. According to Spanoukadis and Zisman [29], inconsistency management can be viewed as a process composed of six activities: detection of overlaps between software artifacts, detection of inconsistencies, diagnosis of inconsistencies, handling of inconsistencies, history tracking of inconsistency management process, and specification of an inconsistency management policy. The activities to be taken depend on the type of inconsistency being addressed [19]. The methods and techniques developed to support inconsistency management activities are based on logics [13, 33], model checking [1, 12], formal frameworks [9, 20, 28], human-centered approaches [4, 26, 33], explicit policies [6], knowledge engineering [35], hypertext [15], abstract dependence graph [2], and static checking [16].

The problem with many existing approaches to traceability and inconsistency management is that they are effective for only a limited portion of the development process, while having little or no support for software product fragments from other parts of the software life cycle [31]. Traceability and inconsistency management support is most frequently found between representations such as formal specifications and program source code that are amenable to automated analysis. No support exists for managing relationships between these representations and less formal representations such as natural language design documents [31].

2.2 The software system build process

In general, during a software development process, software artifacts are either produced directly by developers or by automatic processing tools. This implies a dependency between a source artifact and those directly or indirectly produced from it. Changes in source artifacts may potentially break the dependency relationships and cause the software system out-of-date. Automatic system build tools, such as *make* [7], have to ensure that changes in source artifacts are properly reflected in dependent artifacts. The software build task can be seen as a form of conformance analysis where only the dependency relationships that create automatically derived artifacts are considered.

In practice, system build tools can be optimized by recognizing circumstances where the rebuilding of a software artifact is unnecessary. Different tools will have different build optimization strategies. Gunter [11, 10] has developed a general theory of build optimizations based on a form of abstract interpretation called the *build abstraction*. The theory can be used to characterize the correctness properties of build optimizations and to show how build optimizations can be compared and combined. Gunter bases his model of the dependencies in a system build tool on a theory of concurrent computation over a class of Petri Nets called *production* nets.

Conformance analysis can be seen as a generalization of the build optimization, so this paper generalizes Gunter's formalism to model the conformance analysis process.

3 Representations for software documents and relationships

Software documents and their relationships can be modeled as a network (web) of nodes and links where each node represents a fragment of a document and each link represents a relationship between fragments. While it is common to use graph-based representations for various software engineering processes such as software configuration management [7], it is not common to use the hypertext model. This section discusses the nature of document relationships and describes how we use the hypertext model to represents those relationships to support conformance analysis.

3.1 Document relationships

Relationships among software documents can be divided into *conformance* and *non-conformance* relationships. Conformance relationships represent logical semantic dependencies between documents, such as a requirement "motivating" a design feature. Non-conformance relationships support navigational and organizational tasks, such as "next document" links on individual pages. Non-conformance relationships are not directly relevant to conformance analysis and their maintenance can often be automated using standard Web site management tools.

Conformance relationships can be further divided into the *causal* and *non-causal* classes. Causal conformance relationships (*causal relationships* for short) represent that relationships that carry with them an implied logical ordering of the documents involved. For example, testing and bug reports cannot be produced until an implementation is available, and while it is not necessarily the case that requirement documents will be written before designs, there is certainly a logical relationship between them that makes design depend on requirements. In conformance analysis, the most important characteristic of causal relationships is that causation establishes a partial ordering in time among entities [30]. Therefore, when this partial order in time is violated, it is possible that the conformance between documents has been broken. A causal relationship can be considered as a *relation* between entities: something happens and causes something else to happen. The causal relationship is *transitive*, *irreflexive*, and *antisymmetric*.

Non-causal conformance relationships (*non-causal relationships* for short) exist when documents or parts of them must agree with each other, but the causality cannot be clearly identified. For example, multiple versions of the same document in different languages must agree, but there need not be a causal relationship among them. Since changes to any of the entities in a non-causal relationship may cause the logical semantics of the relationship to become invalid, conformance analysis must be able to account for such relationships. Mathematically, the non-causal relationship is *transitive*, *reflexive*, and *symmetric*.

Regardless of class, document relationships also have variable arity and can have fine granularity.

Variable arity: While simple binary relationships are common in software documents, it is also common for relationships to connect many entities. An entity can have multiple causes with none of them alone sufficient to cause it. It can in turn produce multiple effects. So, causal relationships may have multiple sources and multiple targets. For instance, several items listed in requirement specification A could "affect" the design of an algorithm B. The same pattern holds true for non-causal relationship example, where documents in several languages may need to agree.

Fine granularity: Document relationships can exist both between documents and within a single document. In software documents, which can be rather lengthy, it will be common that relationships will connect relatively small sections of material, such as functions, paragraphs, or subsections. So, document relationships will be fine-grained.

In summary, software document relationships can be divided into three broad classes (non-conformance, causal conformance, and non-causal conformance), have variable arity, and connect fine-grained entities within documents.

3.2 The hypertext model

Because conformance analysis is performed in the domain of linked documents, it is natural to use the *hypertext model* [3] as a basis for representing these relationships.

Though different hypertext systems have variations of the notion, the hypertext model can be defined as a set of intellectual *works* and their inter- and intra-work relationships, represented by *links*. A *work* is an artifact that can be drawn from any medium, such as text, image, or video.[1] A wide variety of terms have been used to describe a work in hypertext systems including *document*, *card*, *frame*, *node*, *object*, and *component*. In the hypertext model, a *link* (or *hyperlink*) is a first-class entity and defined as an association among a set of works or anchors. Although the notion of *anchor* varies in systems, anchors always denote regions of interest within a work and form the endpoints for links.

In the hypertext model, links have two important properties: *arity* and *directionality*. The arity of a link specifies the number of its endpoints. Many hypertext representations, such as HTML, only support binary (two endpoint) links. More sophisticated hypertext representations permit *n-ary* links, which can have a variable numbers of endpoints and have also been called multi-links or multiheaded links.

Link directionality takes two forms: navigational and logical. Navigational directionality refers to the direction(s) in which a link may be traversed. Logical directionality is a semantic quality that is independent of how a link can be traversed. For example, the logical direction in a link that represents the relationship "A causes B" is from the source of the cause, A, to the destination B. Logical directionality has not been widely discussed in the hypertext literature.

3.3 Using hypertext to represent document relationships

We envision using the hypertext model as the basis for representing software documents and their relationships. Each software document will be a *work* in the sense used

[1]When hypertext is extended to support media beyond text, it is sometimes called *hypermedia*, but in current usage, the term "hypertext" is considered to cover all media.

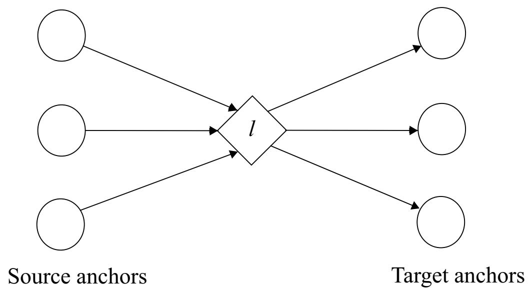

Figure 1. Causal relationship representation

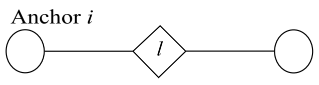

Figure 2. Non-causal relationship representation

by the hypertext model. Software documents will have internal structure, such as that provided by XML, and it will be possible to define an anchor corresponding to any well-defined structural unit.

Links will be divided into three classes corresponding to the three classes of document relationships identified earlier: non-conformance, causal, and non-causal. All links are fundamentally n-ary, though in practice, we might see many binary links. All links may have named types (such as “requires,” “must agree,” or “next item”), but the set of types is not fixed, so that a variety of software processes can be supported.

Non-conformance links are intended to support navigational and organizational relationships that have little or no relevance to determine agreement between documents, such as table of contents and index links. Their navigational directionality is user-specifiable. They do not participate in the conformance analysis process. Each non-conformance link has an identifier and an extensible set of attributes.

Each causal link represents a causal relationship, R. Causal links are always directional, connecting a set of source anchors and a set of target anchors. The source and target sets of a link l are denoted by I_l and O_l , respectively. If a link’s source and target sets each have one element, the link is binary. The directionality of the link is used to represent the causal semantic dependency, but the navigational directionality is user-specifiable. Figure 1 shows a graphical representation of a causal link. In Figure 1, circles represent anchors and the diamond represents a link. Directed edges connect the source anchors to the link and connect the link to its target anchors.

Each non-causal link represents a non-causal relationship, designated by the symbol r. Non-causal links participate in conformance analysis and are logically non-directional. Their navigational directionality can be specified by the user. Let us use r_l to denote the set of nodes connected by the non-causal link l. Figure 2 illustrates this representation, where an undirected edge connects each anchor to a non-causal link.

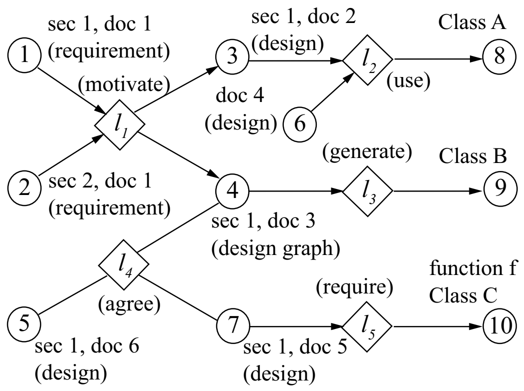

Figure 3. An example of a conformance graph

4 Conformance Analysis

The full set of software documents for a project and all the links among them form a general hypertext graph. An important subgraph is the *conformance graph*, which contains only the conformance links and the nodes connected by them.

4.1 Conformance graph

The conformance graph is a five-tuple

$$G = \{N, L_C, L_N, E_D, E_N\}$$

where N is a finite set of document nodes, L_C is a finite set of causal link nodes, L_N is a finite set of non-causal (conformance) link nodes, E_D is a finite set of directed edges and E_N is a finite set of non-directed edges.

Each node in N represents a document node. Each node in L_C represents a causal link. Each node in L_N represents a non-causal link. Each directed edge in E_D either connects a source anchor to a causal link or a causal link to a target anchor (Figure 1). Each non-directed edge in E_N connects an anchor to a non-causal link (Figure 2). Let $L = L_C \cup L_N$. It is important each relationship is represented by a node, rather than by an edge, as is often the case in other graph representations of dependencies.

Figure 3 shows an example of a conformance graph. There are ten document nodes in N. The links l_1, l_2, l_3,

l_5 are causal links, which are all in L_C. L_N contains only l_4. Causal links have incoming and outgoing edges that are directed, while l_4 and its anchors are connected by non-directed edges.

The conformance graph must be acyclic with respect to conformance links, because cycles would imply that a node causes itself. It must also satisfy the unique producer condition that Gunter defined for the p-net [11]. Conformance links may not be degenerate: 1) Non-causal links must connect at least two anchors, and 2) Causal links must have non-empty source and target sets. There are additional restrictions on the conformance graph when structural relationships among nodes are present [18].

4.2 The timestamp strategy

Our research has developed a method [17] to partially automate the detection of conformance problems using timestamps on links and anchors in a manner similar to that used by the *make* [7] build optimization tool. We assume that both anchors and links are first-class objects that are under version control. New versions of anchors are created each time the material within an anchor is altered in some way. Link versions change when the set of anchors connected by a link is altered. Both anchors and links have unique IDs and version information including version ID, version creator, and a modification timestamp. In addition, links must have a validation timestamp that records the time when the link was last validated by human inspection. When a link gets updated to a new version, the validation timestamp is copied from the previous version.

4.3 Conformance ratings

Our conformance analysis method compares anchor modification timestamps and link validation timestamps to produce a heuristic *conformance rating* of the likely seriousness of conformance problems. Conformance ratings are made on a scale from zero (likely conformance) to ten (likely non-conformance) and have yet to be validated empirically. The conformance rating for a link is determined in one of two ways, depending on whether the link is causal or non-causal.

For causal links, four timestamps are relevant: $t_{src_{max}}$, the maximum version timestamp of the source anchors; $t_{dest_{min}}$, the minimum version timestamp of the destination anchors; $t_{dest_{max}}$, the maximum version timestamp of the destination anchors; and t_{valid}, the validation timestamp of the link. A case analysis of the order of these timestamps has been made, defining conformance ratings for each case, as shown in Table 1. For example, if $t_{src_{max}} \leq t_{dest_{min}}$ and $t_{dest_{max}} \leq t_{valid}$ then the destinations are newer than the sources and the link relationship was validated by a human sometime after the last changes to both. So, the rating is zero. In contrast, if $t_{src_{max}} \leq t_{valid} < t_{dest_{max}}$ then one of the destinations has changed since the last validation and the rating is five. We have arrived at these ratings only by human judgement. A correct and useful formula for these ratings remains an open question and will probably require empirical research.

When causality is not available to provide clues, we turn to a more formulaic approach. Assume that a non-causal link connects N anchors and let t_i be the timestamp of the i^{th} anchor. Then, if there exists any t_i such that $t_i > t_{valid}$ (where n is the number of such anchors) then

$$D = \alpha + (n/N) \times (10 - \alpha)$$

Otherwise, $D = 0$. α is an adjustable parameter representing the minimum rating when the timestamps suggest that a problem exists.

5 Conformance model

The previously described scheme is best suited to informal documents written in natural languages and for multimedia documentation containing images, videos, audios or graphic drawings. For formal documents such as program sources or formal specifications such as Z documents [24, p. 699], UML diagrams [27], ER diagrams [24, p. 305], flow charts, a more automated method could reduce the workload for developers in validating documents and their logical relationships. Fully computerized methods are feasible for formal documents since they are written in formal languages that have clearly defined semantics and syntactic rules. However, it is impossible to provide all possible semantic-dependent methods to do conformance analysis for software documents since developers may have different software development methodologies and engineering tools. Instead, an model that is capable of describing conformance analysis strategies and allows developers to compare their efficiency is more desirable. The model could also help developers to understand the correctness of strategies individually and in combination.

With similar objectives, Gunter [11] has developed an abstract model of dependencies between software configuration items in a build process, based on a theory of concurrent computation over a Petri net. He developed a general theory of build optimizations and used it to show how correctness properties can be characterized and proven, and how build optimizations can be compared. To apply the formal techniques described by Gunter, our research attempts to generalize them from the Petri net model to the hypertext model. The remainder of this section describes our *conformance model* for conformance analysis, which is generalized from Gunter's build model.

Time Relationship	Conformance Rating	Comments
$t_{src_{max}} \leq t_{dest_{min}}$ and $t_{dest_{max}} \leq t_{valid}$	0	Full conformance: dest is newer than src and relationship is validated
$t_{dest_{max}} < t_{src_{max}} \leq t_{valid}$	1	Near full conformance: src is newer than dest, relationship has been validated
$t_{src_{max}} \leq t_{valid} < t_{dest_{max}}$	5	Possible problem: one of destinations has changed since last validation
$t_{valid} < t_{src_{max}}$ t_{dest} undetermined	10	Likely problem: src has changed and relationship is not validated

Table 1. Conformance ratings for causal links based on various time relationships between the anchors and the validation of the link.

5.1 Introduction

Our conformance model explains how the hypertext model can be used in conformance analysis modeling. In this section, the conformance graph, consisting of nodes and links, is referred to as a *hypertext network*. The conformance model is based on three key concepts: *interpretations*, *states*, and *operations*.

An interpretation is a way of viewing a hypertext network. For our purposes, there are two kinds of interpretations of a hypertext of documents and relationships.

- The *standard interpretation* is a distinguished way of viewing the hypertext network in which an anchor node (node, for short) represents the contents of a structural unit of a software document, and a link represents the semantics of the relationship between the nodes that it connects. The standard interpretation is a formal notion designed to represent the natural semantics of software documents. There is only one standard interpretation.

- An *abstract interpretation* is any alternative view of the hypertext network. For conformance analysis, there will be at least one abstraction that provides information useful to the conformance analysis process, which is called a *conformance analysis abstract interpretation*. In this section, since we discuss conformance analysis, let us call it an *abstraction* for short.

A *state* is a binding of some values in some domains to nodes and links. A state in the standard interpretation assigns the contents of fragments in documents to the corresponding nodes and the semantics of relationships to links. A state in an abstraction is called an *abstract state*, which assigns some values (called *abstract values*) to nodes and links for the purpose of detecting conformance problems. Abstract values for links are important as shown in the timestamp strategy. They are not supported in Gunter's build model.

An *operation* is an action by some agent that changes the states of the interpretations. A special *fix* operation of the standard interpretation refers to modifications made to the hypertext network in order to make documents conformant to each other with respect to a relationship. For each abstraction, there is a special *update* operation which will change abstraction values of the hypertext network for the purpose of tracking possible conformance problems.

An example of an abstraction is a timestamp abstraction. For instance, a timestamp abstraction contains assignments of timestamps (or dates) to nodes in the hypertext network. The timestamp abstraction says that if the most recent modification timestamps of sources are older than the modification timestamps of the targets, and all modification timestamps of the targets are older than the link's validation timestamp, then conformance is reached. When developers change the content of a node, the current states in the standard interpretation and abstractions also change.

Let us revisit the example of a conformance graph in Figure 3 and the timestamp abstraction described earlier. Assume that the documents in a project X has been created and their relationships are connected as in the figure. There are ten nodes and five relationships. The link l_4 is a non-causal conformance link while all others are causal links. For example, section 1 of the requirement document $doc1$ and section 2 of $doc1$ “motivate” the designs of features described in the section 1 of design document $doc2$ and in a design graph of $doc3$. The standard state consists of an assignment of content to document nodes and links. The timestamp abstraction has the abstract state that is an assignment of the latest modification timestamps to the document nodes and of latest validation timestamps to links.

Let us denote the timestamp for any node i by t_i, and the timestamp for any link l_j by t_{l_j}. Suppose that the graph has no conformance problems and that the timestamps are such that all conformance ratings are zero. This means that for all causal links $t_{src} \leq t_{dest} \leq t_{valid}$ and for the non-causal link, l_4, $t_i \leq t_{l_4}$ for all nodes i connected by l_4.

Suppose that now, section 1 of document 5 is changed. In this case, the abstract value of node 7 is also changed by the update operation ($t_7^{'}$) to reflect the most recent modification timestamp. Now, $t_7^{'}$ is greater than t_4, t_5, and t_{10}. If a conformance analysis is requested, the abstract values (timestamps) of related nodes are examined. According to the timestamp abstraction, the fact that the source of the causal relationship l_5 (node 7) is newer than its destination (node 10) is an indication that node 7 and node 10 might not be in agreement with each other with respect to the semantics of the link l_5. That is, the design choice in section 1 of document 5 might not require the use of the function f of the Class C anymore. Similarly, nodes 4,5, and 7 might not be in agreement anymore. These points of possible non-conformance will be brought to the developers' attention. Suppose that after examining the contents of nodes, developers make some changes to documents to regain the conformance. These changes cause the change of the standard state. This action is modeled by the *fix* operation. Note that the fix operation can be done by any agent, which could be an automatic tool. Then, after the fix operation is invoked, the *update* operation is also invoked to update timestamps for nodes and links in the hypertext network.

Having described the graphical notations for causal and non-causal links in a conformance graph (in section 4) and the general concepts of the abstraction model, we now introduce a formalism for conformance analysis. Consistency and conformance will be used interchangeably.

This formalism is used to define a concept of conformance for the entire graph, to show how correctness properties of a conformance analysis strategy can be characterized and proved, and to define how changes to the nodes affect the conformance of the graph. It is also used to show the correctness properties of a combination of any conformance analysis strategies. It is important to understand that this formalism says nothing about how the consistency of individual links will be determined. It is presumed that, for each interpretation of the graph, either the standard interpretation or some abstraction, there is a method that can be applied to a link to determine whether or not it is consistent. In the standard interpretation, this method is likely to be an inspection of the link's anchors by a software engineer.

5.2 Formalism

5.2.1 Important concepts

The transitive and reflexive closure of edges corresponding to causal links defines a partial order set on a subset of nodes. The minimal node elements of this partial order set are called *source nodes*.

Let N and L be the set of nodes and links respectively in the conformance graph. Remember that R and r are a causal relation and a non-causal relation, respectively. I_l is the source set and O_l is the target set of anchors for a causal link l. r_l is the set of anchors of a non-causal link l.

Left-closed: A subset X of $N \cup L$ is *left-closed* when $x \in X$ and either $R(y,x)$ or $r(y,x)$, then $y \in X$. In other words, a subset X of the hypertext network is left-closed if for every node x in X, any nodes related to x by a non-causal conformance relationship directly or indirectly, and any nodes that are the causes of x directly or indirectly, also belong to X. Note that a left-closed subset does not have any incoming arcs whose source is outside the set.

Target-closed: A subset X of $N \cup L$ is *target-closed* if, for every causal link $l \in L$, $O_l \cap X \neq \emptyset$ implies $O_l \subseteq X$. That is, X is target-closed when for every causal link l in X, X either contains all of target nodes of l or none of them.

Marking: A *marking* M on a hypertext network is a left-closed and target-closed subset of $N \cup L$. The restriction that M is left-closed and target-closed means that for every causal link l, M either contains all of its target nodes or none of them (target-closed) and that M has no incoming links (left-closed). This notion is used to define the operational semantics in the next section. The idea behind the definition of a target-closed subset is that during the conformance analysis process, a causal link either has not been examined or has been examined at the same time with all of its targets.

Indexed family of sets: An *indexed family of sets*, S, is an indexing collection $\mathcal{I}$ together with a function associating each element $i \in \mathcal{I}$ with a set S_i. S is denoted by $(S_i \mid i \in \mathcal{I})$. An *assignment function* of such an indexed family of sets $S = (S_i \mid i \in \mathcal{I})$ is a function associating with each $i \in \mathcal{I}$, an element $s_i \in S_i$. It is denoted by $s = (s_i \mid i \in \mathcal{I})$. The *product* of S is defined as the set of all possible assignment functions of S. It is denoted by $\prod(S_i \mid i \in \mathcal{I})$.

5.2.2 States

For $n \in N$, V_n is defined as the domain of the values of the node n. For the standard interpretation, V_n is the set of strings representing the content of the node n. For abstractions, V_n is the domain of abstract values. This domain is chosen precisely because its values can be used to detect non-conformance.

For the standard interpretation:

- If l is a causal link, the value of l is the causal relation between the values of nodes in I_l and those of nodes in O_l.
- If l is a non-causal link, the value of l is the relation between the values of nodes in r_l.

For abstractions:

For $l \in L$, let v_l be the abstract value of the link l and V_l be the domain of these abstract values. For each $l \in L$, let c_l be the constraint relation on abstract values of nodes connected by l and the abstract value of l. The purpose of c_l is to detect conformance problems. Let us denote the domain of c_l by C_l. The definition of C_l is as follows:

- If l is a causal link, then $C_l \subseteq \prod(V_n \mid n \in I_l) \times \prod(V_n \mid n \in O_l) \times \prod(V_k \mid k \in \{l\})$. The constraint relation of l is a relation between the assignment function assigning abstract values to nodes in I_l, the assignment function assigning abstract values to nodes in O_l, and the assignment function assigning an abstract value to the link l.

- If l is a non-causal link, $C_l \subseteq \prod(V_n \mid n \in r_l) \times \prod(V_k \mid k \in \{l\})$. The constraint relation of l is a relation between the assignment function assigning abstract values to nodes in r_l and the assignment function assigning an abstract value to the link l.

A *state* in an interpretation is 3-tuple of an assignment function of $(V_n \mid n \in N)$, an assignment function of $(V_l \mid l \in L)$, and an assignment function of $(C_l \mid l \in L)$.

5.2.3 Consistent links

In the standard interpretation, a link l is said to be *consistent* in a state s if the nodes connected by l are conformant to each other with respect to the natural semantics of l. For abstractions:

For causal links: A link l is said to be *consistent* in state s of an interpretation B if and only if the relation $C_l((s_i \mid i \in I_l), (s_i \mid i \in O_l), (s_k \mid k \in \{l\}))$ is true. It is denoted by the consistency predicate $s \vdash_B l$. In other words, state s in the context of an interpretation B entails the consistency of the link l.

For non-causal links: A link l is said to be *consistent* in state s of an interpretation B if and only if the relation $C_l((s_i \mid i \in r_l), (s_k \mid k \in \{l\}))$ is true.

A left-closed subset, M, of the hypertext network is said to be *consistent* with respect to s, if every causal link in M that has a target in M and every non-causal link in M that has all anchors in M are consistent.

For example, let us consider a timestamp abstraction B for causal links. $V_n = \{0, 1, 2, 3, ...\}$ for all nodes n. $V_l = \{0, 1, 2, 3, ...\}$ for all links l. For each link l and the state t of B, $t \vdash_B l$ holds if $\max\{t_n \mid n \in I_l\} \leq \min\{t_m \mid m \in O_l\}$ and $\max\{t_m \mid m \in O_l\} \leq t_l$. That is, a relationship is considered consistent in this timestamp abstraction if the maximum of timestamps of its sources is less than or equal to the minimum of those of its targets and all of the timestamps of its targets are less than or equal to the validation timestamp of the relationship l.

5.2.4 Relationship between the standard interpretation and an abstraction

Suppose that s and t are states of the standard interpretation A and an abstract interpretation B, respectively. The changing of the states s in the standard interpretation reflects the developers' modifications to software documents. That is, the state s is in the "real content" level. In contrast, the changing of states t reflects the changes of abstract values for conformance tracking. In order for B to be a correct abstract interpretation of A, a relation must be defined between those two changes of s and t. In other words, we need to have a relation between states s of A and states t of B such that the relation holds when t is viewed as a correct abstraction of s. That relation is called the abstraction relation $\Phi : A \rightarrow B$ in Gunter's build model. Our research adapts this notion to the hypertext model. The following soundness properties formally define the relation Φ for hypertext. The abstraction relation is a relation between states in the standard interpretation A and states in an abstraction B, satisfying the conformance soundness and deletion soundness properties:

Property 1 (Conformance soundness) *If $\Phi(s, t)$ and $t \vdash_B l$ then $s \vdash_A l$. That is, if B says that link l is consistent, then the corresponding state in A is consistent for the link l too.*

It is certain that this property is necessary since the purpose of the abstraction B is to provide an indication of a link's consistency. Before defining deletion soundness, we need to define a new notion. Given a function $f : X \rightarrow Y$ and a subset U of the domain of f. Let us denote the *restriction of* f to U by $f \mid U$.

Property 2 (Deletion soundness) *If $\Phi(s, t)$ then $\Phi(s \mid U, t \mid U)$ for any subset $U \subseteq N$.*

This property defines the soundness with respect to any deletion of nodes. It is a natural requirement since any software document could be removed from the system.

5.2.5 Operation *"fix"* in the standard interpretation

Before formally defining the fix operation, we need to define the *change set* notion. In build model, the change set is defined as a subset U of the node set N such that either 1) each element of U is a source node or 2) there is a link l such that $U = O_l$. That is, in the build process, there are three types of changes: a source file is modified; target files are produced; or a node is deleted. In the build process, only source nodes are produced or modified by human beings. In contrast, in conformance analysis, both sources and targets can be produced or modified by human beings, so any node can be in a change set. Therefore, the "fix" operation must consider a general change set U, while the "build server" in Gunter's build model did not.

The "fix" operation of the standard interpretation is used to correct an *inconsistent* link between nodes bridged by that link. Let us denote the "fix" operation by α. Function α takes as its arguments a link l, a state s in the standard interpretation A and return a change set U and a new state s' in A such that the following conditions are satisfied:

1. The link l is consistent in the new state s' (i.e., $s' \vdash_A l$),
2. If l is a causal link, $\neg\exists l' \in L$ such that $s \vdash_A l'$, $s' \not\vdash_A l'$ and l' is connected to some node in I_l, and
3. If l is a non-causal link, $\neg\exists l' \in L$ such that $s \vdash_A l'$, $s' \not\vdash_A l'$ and l' is connected to some node in r_l.

The first condition guarantees that after fixing, the link l is consistent in the new state. The second and the third conditions require that the fixing changes must not break the consistency of some neighboring links. This restriction is important to the soundness of the conformance analysis that will be described later. It is important to notice that the fix operation corrects one link at a time, together with connected nodes.

In the build process, a rebuilding for a dependency relationship between source files and the files that they generate is a fix operation α. However, rebuilding does not change the source files. Rebuilding only regenerates the target files if necessary. Therefore, to model a build process, α must change only the target files of the dependency relationship.

5.2.6 Operation "update" for abstraction relation Φ

The purpose of the update operation is to record the new abstraction values of the current state in an abstract interpretation after the fix operation is invoked. The operational scenario for "fix" and "update" is as follows. Assume that s and t are states in the standard interpretation A and an abstraction B such that t is a correct abstraction of s (i.e., $\Phi(s,t)$ holds). Then, changes are made to some document nodes and s changes into s''. The update operation is then invoked so that t changes into t'' and new abstraction values are recorded. Suppose that document nodes are no longer in agreement with each other with respect to a link. When conformance analysis is requested, abstraction B says that non-conformance occurs in t''. The fix operation is invoked to correct the inconsistency of the link, in the standard intepretation, changing s'' into s'. Then, the update operation is invoked so that t'' changes into t' and new abstraction values are recorded. The nodes that have been changed by the fix operation are in the change set U.

The update function, β takes three arguments: 1) a change set U, 2) a state in standard interpretation s', and 3) an abstract state t. It returns a new abstract state t'.

The function β must satisfy the following property:

Property 3 (Abstraction) *If $s \mid (N \setminus U) = s' \mid (N \setminus U)$ and t is a correct abstraction of s (i.e., $\Phi(s,t)$ holds), then $\Phi(s', \beta(U, s', t))$ holds.*

That is, after being updated, the new abstract state t' is also a correct abstraction of s'. This property is important to β since the new abstraction values need to be updated correctly. For example, in a timestamp abstraction, the function β is defined as follows:

- For all nodes in the set $N \setminus U, \beta(U, s', t) = t$.
- For a node $x \in U, \beta(U, s', t) = 1+\max\{t_x \mid x \in C_x\}$ with $C_x = \{y \mid R(y,x)$ or $R(x,y)$ or $r(x,y)\}$.

The first rule says that there is no need to update abstract values for nodes that are not changed. The second rule records the timestamp for a changed node x such that the timestamp is greater than timestamps of any nodes related to x by any links.

5.2.7 Operational semantics

Operational rules represent the semantic operations of the hypertext model in which the states are changed and operations are invoked. They are defined based on the *marking* concept. Remember that a marking M on a hypertext network is a left-closed and target-closed subset of $N \cup L$. A marking M is said to be *consistent* with respect to a state s, if every causal link in M that has a target in M and every non-causal link in M that has all anchors in M are consistent. For build optimization, when some source node is changed, it is placed in a marking. The set of documents is made consistent by applying a series of operations, link by link, that make additional documents consistent with the change and adds those updated documents to the marking. This process of marking continues until no more documents can be marked. For conformance analysis, the process of marking is more complex than that of Gunter's build model due to the presence of undirected edges of non-causal links in a conformance graph.

First of all, a few notions need to be defined:

- Relative to a marking M, a causal link l has three possible states:
  - M does not contain l or its targets. Let us denote this state by (M/l).
  - M contains the link l and its sources but none of its targets. That is, $l \in M$ and $O_l \cap M = \emptyset$. Let us denote this state by (l/M).
  - M contains all sources, targets and the link l. That is, $O_l \subseteq M$. Let us denote this state by $(l\backslash M)$.

- Similarly, relative to a marking M, a non-causal link l has three possible states:
  - M does not contain l. Let us denote this state by (M/l).
  - M contains the link l and at least one (but not all) of its anchor nodes. That is, $l \in M$, $r_l \cap M \neq \emptyset$, and $r_l \nsubseteq M$. Let us denote this state by (l/M).
  - M contains the link l and all its anchor nodes. That is, $l \in M$ and $r_l \subseteq M$. Let us denote this state by $(l \backslash M)$.

Note that all other states are excluded by the left-closed and target-closed properties of a marking M.

Operational rules: Let us denote the evaluation relation by $\rightarrow$. An *evaluation state* is a triple consisting of a marking M, a standard state s and an abstract state t. It is denoted by (M, s, t). Let us denote the initial evaluation state by (M_0, s_0, t_0). M_0 consists of "source" nodes, i.e., the nodes that are not in the target set of any link. The initial s_0 contains assignments of the contents of documents to nodes.

Assume that the abstraction relation is Φ. Its "update" function β is invoked at this point on these initial source nodes to create an abstraction state t_0 that satisfies the conformance and deletion soundness conditions of Φ. Then, the following rules of operation semantics formally define the process of marking discussed previously:

1. $\dfrac{(M/l) \quad t \nvdash_B l}{(M, s, t) \rightarrow (M \cup \{l\}, s, t)}$ [Initiation-causal]
2. $\dfrac{(l/M) \quad (s', U) = \alpha(l, s)}{(M, s, t) \rightarrow (M \cup O_l, s', \beta(U, s', t))}$ [Execution-causal]
3. $\dfrac{(M/l) \quad t \vdash_B l}{(M, s, t) \rightarrow (M \cup \{l\} \cup O_l, s, t)}$ [Omission-causal]

(Note: U is the change set)

The initiation rule for causal links says that if the state t in the context of an abstraction B decides that l is not consistent, the marking M continues to spread out to include the link. The state of l relative to M now changes to (l/M).

The execution rule for causal links says that when conformance analysis is requested and the marking M includes the link l, the fix operation α is invoked to change s into s', then M spreads out to include the target set of l and the update operation β is invoked to update the abstraction values.

The omission rule for causal links says that if the state t in the context of an abstraction B decides that l is consistent, (i.e., the fix operation α does not need to be invoked when conformance analysis is requested), M spreads out to include both the link and its target set.

Similarly, the operational semantic rules for non-causal links are:

1. $\dfrac{(M/l) \quad t \nvdash_B l}{(M, s, t) \rightarrow (M \cup \{l\}, s, t)}$ [Initiation-non-causal]
2. $\dfrac{(l/M) \quad (s', U) = \alpha(l, s)}{(M, s, t) \rightarrow (M \cup r_l, s', \beta(U, s', t))}$ [Execution-non-causal]
3. $\dfrac{(M/l) \quad t \vdash_B l}{(M, s, t) \rightarrow (M \cup \{l\} \cup r_l, s, t)}$ [Omission-non-causal]

5.2.8 Soundness theorem of the marking process

As previously described, conformance analysis is modeled based on the marking process. In order for conformance analysis to be sound, the process of marking must be sound. A formal definition of the soundness of the marking process is needed. The formal definition is inspired by the soundness theorem in type systems described by Pierce [21].

First, the concepts of *terminate* and *stuck* must be defined. The marking process is said to *terminate* if the initial marking M_0 continues to spread out and finally marks all nodes and links in the hypertext network. The process is said to become *stuck* if at a point different than terminate, there are no operational rules that can be applied.

The soundness is based on two properties: the process does not become stuck and the consistency of relationships in the marking is always preserved. Formally, soundness theorem is expressed as two properties:

Property 4 (Progress) *A consistent marking never becomes stuck. Either it terminates or it can take a step according to the evaluation rules in the operational semantics.*

Property 5 (Preservation) *Let A be the standard interpretation and B be an abstraction. If M is a consistent marking, $(M, s, t) \rightarrow (M', s', t')$ with respect to a fix operation α of A, and Φ is an abstraction relation from A to B with its update operation β, then M' is also a consistent marking in s', and t' is also a correct abstraction of s' (*i.e. $\Phi(s', t')$ *holds).*

These properties guarantees the process finally ends at a "terminate" consistent marking including all nodes and links. The proof of this theorem has been developed.

5.2.9 Combining multiple abstractions

Conformance analysis for formal documents can be automated depending on the built-in knowledge about the semantics of those documents. In other words, software development environments (SDEs) can provide semantics-dependent abstractions for the different types of formal documents that they support. For informal documents such as multimedia documentation, a variety of technologies such as natural language processing (NLP) technologies, information retrieval technologies, image and speech recognition technologies could be employed to derive relationships, dependencies and to create conformance analysis abstractions. However, limitations on practical applications of NLP and recognition technologies prevent them from being used as the sole technique in a conformance analysis. In contrast, the timestamp abstraction presented earlier is general, lightweight and can be used in combination with other techniques. The practical question is how techniques from different abstractions can be combined to yield an overall analysis process taking advantages of all abstractions together. A generalized semantics of the conformance analysis process that allows for multiple abstractions is required.

Let us call A the standard interpretation. Let $\Phi_i : A \rightarrow B_i$ be the i^{th} abstraction relation ($i = 1...n$). Suppose that s is a standard state and t_i is the state of the abstract interpretation B_i such that $\Phi_i(s, t_i)$ holds. The evaluation state in operational rules is in the form of $(M, s, t_1, ..., t_n)$. Let the update operations be β_i.

The new operational semantics are given in Figure 4.

The first rule says that if all abstractions B_i decide that l is not consistent in their states t_i, the marking M continues to spread out to include the link. The second one says that when conformance analysis is requested and the marking M includes the link l, the fix operation α is invoked to change s into s', then M spreads out to include the target set of l and the update operation β is also invoked on changed nodes to update the abstraction values for all abstractions. The last one says that if at least one abstraction B_i decides that l is consistent in its state t_i (i.e., according to that abstraction B_i, the fix operation does not need to be invoked), M spreads out to include both the link and its target set. Except that abstraction, other abstractions update their values using their update operations.

Similarly, operation semantics could be extended for non-causal links in the case of multiple abstractions. The soundness property could be developed to accommodate the multiple abstractions in a similar way.

6 Conclusions and Future Work

In this paper, we have described the conformance problem for software documents, discussed its relationship with other software engineering problems, developed a timestamp strategy and a formal model of the conformance analysis process. The model can be used as a mathematical tool to determine the soundness of a conformance analysis strategy. Developers can model a conformance analysis strategy by defining an abstraction. To define an abstraction, they must define abstract values and constraint relations for nodes and links in the hypertext network, the abstraction relation Φ, and the update function β. The soundness theorem can be used to ensure that the conformance abstraction is sound when the abstraction is defined properly. The model also provides a framework for identifying valuable research directions. Researchers can try to improve conformance analysis by identifying new abstractions that better represent the semantics of documents or they can improve the analysis techniques for an existing abstraction.

Developers can use the generalized formalism to verify the soundness of their combined strategy such as a combination between a semantics-dependent abstraction supported by a SDE for formal documents and the timestamp abstraction described earlier. A related question that might arise when multiple abstractions are discussed is whether one conformance analysis abstraction is “weaker” than another one. When multiple abstractions are used in conformance analysis, one abstraction might correctly detect conformance while a weaker one did not. Principles for comparing conformance analysis strategies are being investigated. Furthermore, we will be applying the conformance model as we build prototype tools for conformance analysis in the Software Concordance environment. Our current efforts are focused on the creation of a versioned hypermedia infrastructure suitable for conformance analysis. Other research is developing a software document testbed and planned research will explore issues to support document evolution, consistency, and change management.

References

[1] W. Chan, R. Anderson, P. Beame, S. Burns, F. Modugno, D. Notkin, and J. Reese. Model checking large software specifications. *IEEE Transactions on Software Engineering*, 24(7):498–520, 1998.

[2] K. Chen and V. Rajich. Ripples: tool for change in legacy software. In *Proceedings of the 2001 IEEE International Conference on Software Maintenance*, pages 230–239. IEEE, 2001.

[3] J. Conklin. Hypertext: An introduction and survey. *IEEE Computer*, 20(9):17–41, 1987.

[4] G. Cugola, E. D. Nitto, A. Fuggetta, and C. Ghezzi. A framework for formalizing inconsistencies and deviations in human-centered systems. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 5(3):191–230, 1996.

[5] J. Dick. Rich traceability. In *Proceedings of the Automated Software Engineering, Traceability Workshop*, 2002.

[6] W. Emmerich, A. Finkelstein, C. Montangero, S. Antonelli, S. Armitage, and R. Stevens. Managing standards compliance. *IEEE Transactions on Software Engineering*, 25(6):836–851, 1999.

[7] S. Feldman. Make: A program for maintaining computer programs. *Software Practice*, 9:255–265, 1979.

[8] O. Gotel and A. Ginkelstein. An analysis of the requirement traceability problem. In *Proceedings of the 1st International Conference on Requirements Engineering (ICRE’94)*, pages 94–102, 1994.

1. $$\frac{(M/l) \quad t_i \nvdash_{B_i} l \; for \; all \; i}{(M, s, t_1, t_2, ..., t_n) \rightarrow (M \cup \{l\}, s, t_1, t_2, ..., t_n)}$$ [Generalized-Initiation-causal]

2. $$\frac{(l/M) \quad (s', U) = \alpha(l, s)}{(M, s, t_1, t_2, ..., t_n) \rightarrow (M \cup O_l, s', \beta_1(U, s', t_1), \beta_2(U, s', t_2), ..., \beta_n(U, s', t_n))}$$ [Generalized-Execution-causal]

3. $$\frac{(M/l) \quad t_i \vdash_{B_i} l \; for \; some \; i}{(M, s, t_1, t_2, ..., t_n) \rightarrow (M \cup \{l\} \cup O_l, s, t'_1, t'_2, ..., t'_n)}$$ [Generalized-Omission-causal]

where $t'_j = t_j$ if $t_j \vdash_{B_j} l$ and $t'_j = \beta_j(U, s', t_j)$ otherwise.

Figure 4. Generalized operational semantics

[9] J. Grundy, J. Hosking, and W. Mugridge. Inconsistency management for multiple-view software development environments. *IEEE Transactions on Software Engineering*, 24(11):960–981, 1998.

[10] C. A. Gunter. Abstracting dependencies between software configuration items. In *Proceedings of the 4th ACM SIGSOFT Symposium on Foundations of Software Engineering*, pages 167–178. ACM Press, 1996.

[11] C. A. Gunter. Abstracting dependencies between software configuration items. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 9(1):94–131, 2000.

[12] C. L. Heitmeyer, R. D. Jeffords, and B. G. Labaw. Automated consistency checking of requirements specifications. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 5(3):231–261, 1996.

[13] A. Hunter and B. Nuseibeh. Managing inconsistent specifications: reasoning, analysis, and action. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 7(4):335–367, 1998.

[14] A. Knethen. Automatic change support based on a trace model. In *Proceedings of the 2002 Automated Software Engineering (ASE 2002), Traceability Workshop*, 2002.

[15] J. Koskinen. Empirical evaluation of hypertext access structures. *Journal of Software Maintenance and Evolution: Research and Practice*, 14(2):83–108, 2002.

[16] C. Nentwich, W. Emmerich, and A. Finkelstein. Static consistency checking for distributed specifications. In *Proceedings of the 2001 Automated Software Engineering (ASE 2001)*, pages 115–124. IEEE, 2001.

[17] T. Nguyen, S. C. Gupta, and E. V. Munson. Versioned hypermedia can improve software document management. In *Proceedings of the Thirteenth Conference on Hypertext and Hypermedia*, pages 192–193. ACM Press, 2002.

[18] T. N. Nguyen. The Software Concordance. University of Wisconsin-Milwaukee Ph. D. Proposal, August 2002.

[19] B. Nuseibeh, S. Easterbrook, and A. Russo. Leveraging inconsistency in software development. *IEEE Computer*, 33(4):24–29, 2000.

[20] B. Nuseibeh, J. Kramer, and A. Finkelstein. A framework for expressing the relationships between multiple views in requirements specification. *IEEE Transactions on Software Engineering*, 20(10):760–773, 1994.

[21] B. C. Pierce. *Type-Theoretic Foundations for Programming Languages*. MIT Press, 2000.

[22] F. Pinheiro and J. Goguen. An Object-Oriented Tool for Tracing Requirements. *IEEE Software*, 13(2):52–64, 1996.

[23] K. Pohl. PRO-ART: Enabling requirements pre-traceability. In *Proceedings of the 2nd International Conference on Requirements Engineering (ICRE'96)*, pages 76–85, 1996.

[24] R. S. Pressman. *Software Engineering: A Practitioner's Approach*. McGraw Hill, 5th edition, June 2000.

[25] B. Ramesh and M. Jarke. Toward reference model for requirements traceability. *IEEE Transactions on Software Engineering*, 27(1):58–93, 2001.

[26] W. Robinson and S. Pawlowski. Managing requirements inconsistency with development goal monitors. *IEEE Transactions on Software Engineering*, 25(6):816–835, 1999.

[27] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual*. Object Technology Series. Addison-Wesley, 1998.

[28] I. Sommerville, P. Sawyer, and S. Viller. Managing process inconsistency using viewpoints. *IEEE Transactions on Software Engineering*, 25(6):784–799, 1999.

[29] G. Spanoudakis and A. Zisman. Inconsistency management in software engineering: Survey and open research issues. *Handbook of Software Engineering and Knowledge Engineering*, 1:24–29, 2001.

[30] P. Spirtes, C. Glymour, and R. Scheines. *Causation, Prediction, and Search*. Adaptive Computation and Machine Learning. The MIT Press, 2000.

[31] D. Strasunskas. Traceability in collaborative systems development from lifecycle perspective — position paper. In *Proceedings of the 2002 Automated Software Engineering (ASE 2002), Traceability Workshop*. IEEE, 2002.

[32] M. Toranzo and J. Castro. A comprehensive traceability model to support the design of interactive systems. In *Proceedings of the ECOOP Workshops 1999, LNCS 1743*, pages 283–284. Springer-Verlag, 1999.

[33] A. van Lamsweerde and E. Letier. Handling obstacles in goal-oriented requirements engineering. *IEEE Transactions on Software Engineering*, 26(10):978–1005, 2000.

[34] R. Watkins and M. Neal. Why and how of requirements tracing. *IEEE Software*, 11(4):104–106, 1994.

[35] A. Zisman and A. Kozlenkov. Knowledge-based approach to consistency management of UML specifications. In *Proceedings of the Automated Software Engineering*, 2001.