When and How to Visualize Traceability Links?

Andrian Marcus, Xinrong Xie, Denys Poshyvanyk
Department of Computer Science
Wayne State University
Detroit MI 48202

{amarcus, xxr, denys}@wayne.edu

ABSTRACT

This position paper discusses the situations when visualizing traceability links is opportune, as well as what information pertaining to these links should be visualized and how.

It also presents a prototype tool, which is used to visualize traceability links to provide support for the user during recovery, maintenance, and browsing of such links.

Categories and Subject Descriptors

D.2.6 [**Software Engineering]**: Programming environments – *Graphical environments, integrated environments, interactive environments, and programmer workbench.*

General Terms

Documentation.

Keywords

Traceability, software visualization

1. INTRODUCTION

Traceability among various software artifacts is an important issue in software engineering as it supports a variety of tasks such as testing, re-documentation, or comprehension. Much work in the area is focused on the recovery of such traceability links between artifacts [2, 7, 13, 15, 16], as in many cases these links are not explicitly represented in the software system. In other cases, parts of the system evolve at different speeds and the existing links are not updated. Other work in the area is focused on the representation [10, 12, 17, 18, 22] and maintenance [1, 4] of traceability links.

The software maintenance research community is developing tools that support users in better understanding and changing existing software. In achieving these goals, such tools provide the user with a wealth of information about the software system. Traceability links are often needed to support such tasks and they should also be used with this type of tools. In order to achieve that, we need to establish in what context and circumstances traceability links should be incorporated into maintenance tools and how they should be presented to the user.

One can argue that representing and navigating traceability links is a trivial matter, as they can be shown as a simple matrix, a graph, or a set of document pairs. In contrast, we advocate in this paper that visualizing traceability links is important, non trivial, and considerable support is needed to recover, browse and maintain these links.

In support of our position, we discuss the elements and properties of the traceability links that lend themselves for visualization. We also propose a set of high level requirements for a general visualization tool that would support browsing of traceability links. Finally, we present a prototype of a software visualization tool we are working on, which implements some of these requirements.

Requirements for visualization of traceability links should result from a discussion in the research community and industry practitioners. This position paper desires to be an opening statement in such a discussion.

2. ELEMENTS AND PROPERTIES OF TRACEABILITY LINKS

When dealing with traceability links, one has to keep track of several things. Traceability links have intrinsic components: source and target. These elements, which indicate various software artifacts, have several properties that are important: artifact name; artifact type (e.g., requirement, design diagram, test case, manual, etc.); location; creation time; update time; version (e.g., the version in CVS for example); etc.

In addition to the properties of the source and the target, a link also has several general properties such as: discovery method (e.g., automatic, semi-automatic, manual, explicit), which indicates how this link was identified; creation time; update time; documentation, which can contain author information and rationale description; version, which also provides a reference to the previous structure of the link that evolved into this current version; usage history, which shows when this particular link has been browsed by a user; status (e.g., active or deleted); etc. Some of these properties are independent, while others are composite (e.g., version depends on update time).

The elements and properties of traceability links determine many categories of such links. Different tasks may require access to a specific set of links, based on their properties. For example, during the maintenance of such links, the user may need to access only those links that may not be consistent anymore (i.e., either their source or target changed). As no two

links are created equal, capturing and presenting this additional information to the user is important.

Some categories are already defined in the literature based on other information. For example, some classifications are based on a meta-model for requirement traceability [17], which defines four types of traceability links: *Satisfies Links*, *Dependency Links*, *Evolves-To Links*, and *Rationale Links*. *Satisfies Links* and *Dependency Links* form a group called *product-related*, which describe properties and relationships of design objects, while the *Evolves-To Links* and the *Rationale Links* belong to a second group called *process-related*, which can be captured only by looking at the history of actions taken in the traceability process itself.

Action-centric views on traceability can be created. For example, traceability links are also captured by Pohl's dependency model [16], eighteen different dependency links are categorized into five classes: *Condition Links*, *Content Links*, *Documents Links*, *Evolutionary Links*, and *Abstraction Links*. *Leads To* and *Modifies* links are defined between requirements and decisions, while *Implies* and *Creates* links exist between decisions and design objects [18].

Traceability data also determines various types of links [6]: *Product Data*, *Supplementary Product*, *Process Observation Data*, and *Dependency Data*, which provides a data-centric view on traceability.

A recovery method-centric view is presented in [5]: *Lost Links*, which are recovered by a recovery tool but not traced by users; *Warning Links*, which are traced by the user but missed by the tool; *False Positive Links*, which are recovered by the tool but classified as false positive by the user; and *Normal Links*, which are recovered by the tool and confirmed by the user.

Probably the most common classification of traceability links is the artifact-centric view, which may be created based on the type of the source and target artifacts. At a high level, one dimension distinguishes between *vertical traceability* and *horizontal traceability* [14]. A second dimension takes into account types of links among items, which can be either *explicit* or *implicit* (this property can be used to generate recovery method-centric view as well). Another dimension divides traceability links into *Structural* and *Cognitive links*.

We want to see in a traceability management tool (which would of course include a visualization component) the means that will allow the user to access and update the link properties and define their own categories (or views) of interest, based on the linkage properties. Each view may be best suited for specific tasks. For example, one can define a view based on how many times the links were updated, used or investigated.

3. WHY AND WHEN DO WE NEED TO VISUALIZE TRACEABILITY LINKS?

There is no standard way to store or represent traceability links. Traditionally, they are stored as a matrix and represented as graphs. While simplicity is the main advantage of the traditional approaches, they are unsuitable for the representation of all the information relevant to the traceability links. In addition to their intrinsic elements, traceability links have several properties (see section 2) and recent research defined several types of traceability links based on various criteria (see section 2). These attributes provide useful information to the user during many engineering or maintenance tasks, so representation of such information in conjunction with the traceability links is quite desirable.

Many software analysis tools developed today to support software evolution are integrated with IDEs, they have a visualization component, and are designed to interface with other similar tools. When information about traceability links is needed while using such a tool, it makes more sense to create a representation that matches the one used by the tool (or IDE) than to use a traditional representation of such links. In such cases, visualizing traceability links will help supporting software analysis and understanding.

One can still argue that sophisticated visualization is still not needed as, given a source artifact, it is easy to simply list all linked artifacts and their properties. While this is true, there are additional scenarios when simple, but more powerful visualization techniques should be used. As discussed in section 2, the properties of traceability links determine several views over the links. Thus, when showing all the links with a given source, we need to differentiate between the multitude of links types, as only some may be of interest to the user. More than that, sometimes developers work with several artifacts at the same time (e.g., parts of the source code) and they need to see all the related artifacts to those under analysis, or an intersection of them. It is thus desirable to display at the same time all the traceability links associated with these artifacts and to provide visual differences between links to or from different artifacts, as well as allow the users to visually filter out elements that are not of interest. Once again, to support such user needs, visualization is needed and it has to be quite powerful (i.e., beyond a simple list of artifacts).

Visualizing different views of traceability, based on the linkage properties, will support the user in solving development and maintenance problems, as well as in gaining a better understanding of the system.

In addition, visualizing traceability links will directly support three main activities related to these links: *recovery*, *browsing* and *maintenance* of traceability links. Browsing, in turn, supports other software engineering tasks, as discussed before.

There is no completely automated traceability recovery process, as they all include a human component. Decisions on the links suggested by a tool must be taken by the user. In these situations, recovery-method centric view may be visualized and help the user with the decision. For example, during the recovery of traceability links, the user may need to know that there are multiple links from a given source artifact and how many of them are false positives.

Finally, traceability links, once recovered, need to be maintained in a consistent way to be useful in other processes or tasks. During the maintenance of the traceability links, developers need to have access to all relevant artifacts and link properties. They also must be able to change these properties and links. Once again, visualization would be a plus in this task.

4. REQUIREMENTS FOR VISUALIZING TREACEABILITY LINKS

Based on the situations that warrant the use of visualization to represent traceability links and on the information that needs to be visualized, we are proposing a set of high level requirements for visualizing traceability links to support their recovery, browsing, and maintenance.

Such a tool should be able to:

1. Visualize and store traceability links among various artifacts, regardless of the extraction methodology used;
2. Interface or integrate with traceability link recovery tools;
3. Allow the user to browse the traceability links through multiple types of user interactions;
4. Allow the user to add, delete and edit the properties of existing links and their connecting artifacts;
5. Seamlessly integrate with an IDE to support a common look and feel for the software artifacts and the traceability links. Changes on one side (i.e., IDE or links) should propagate to the other;
6. Interoperate with other software engineering tools (e.g., analysis tools, document management tools, etc.);
7. Capture and maintain browsing history for traceability links;
8. Provide comprehensive configuration management and change tracking facilities, focused on the link properties and integrated with the source code management tools;
9. Support various data representation formats;
10. Support user querying and filtering of the traceability links;
11. Offer flexible and user customizable views of the traceability data;
12. Analyze and summarize the data on the traceability process and links.

This is of course a non exhaustive list, which we hope to further expand and change (if needed) following discussions with fellow researchers in the community.

5. VISUALIZING AND REPRESENTING TRACEABILITY LINKS IN PRACTICE

Based on these high-level requirements, we developed a prototype tool, called TraceViz (see Figure 2). Some of the requirements are already refined and implemented (partially) in this prototype.

TraceViz is integrated into the Eclipse IDE as a plug-in and its implementation is based on an open-source visualization Eclipse plug-in, Creole [11] – requirement #5. TraceViz is linked with the Eclipse text editor and there is easy navigation between the TraceViz view and the source code.

TraceViz is integrated (in part) with our tool [13], which uses latent semantic indexing to recover traceability links between source code and external documentation – requirement #2. However, the tool could be used in conjunction with any other recovery method.

The traceability link data is stored in a simple XML format, which captures all the elements and properties of the links (see Figure 1 – requirement #1. This format is of course open to debate and modifications. Arguments exist to organize the data based on artifacts, rather than the links, as most software maintenance tools use such a representation. This format is adopted for simplicity at this stage.

```
<Link>
    <Elements>
    <Source>
       <Name> </Name>
       <Artifact type> </Artifact type>
       <Path> </Path>
         …
    </Source>
    <Target>
       <Name> </Name>
       <Artifact type> </Artifact type>
       <Path> </Path>
         …
    </Target>
    </Elements>
    <Attributes>
         …
    </Attributes>
</Link>
```

Figure 1. XML format of the traceability link data

TraceViz has some limited ability to analyze the traceability data. It can use the stored information to extract views, based on user specified values of link attributes – requirements #11 and #12. The current prototype supports the following views :

- Recovery method-based view – groups manually recovered vs. automatically recovered links, as well as the four groups defined in [5] (see section 2);
- Consistency-based view – groups together links whose elements changed or did not change;
- Artifact-based view - groups together links with the target of the same type.

The user can define additional views and categories, based on other attributes, or group of attributes. The data for these categories is saved together with the project in separate files. TraceViz provides two mechanisms to visualize the categories: color and position.

It can handle several document types, through Eclipse, in addition to the source code and it allows the user to add and delete links, as well as to modify their properties manually – requirement #4.

5.1. The TraceViz User Interface

TraceViz uses standard Eclipse views to host the major components of the user interface (UI). The TraceViz UI has three major parts (see Figure 2):

1. The *elements area*, on the left, which contains the source and target browsers;
2. The *link area*, in the middle, which shows the links for a specific source or target (one of them is selected in the

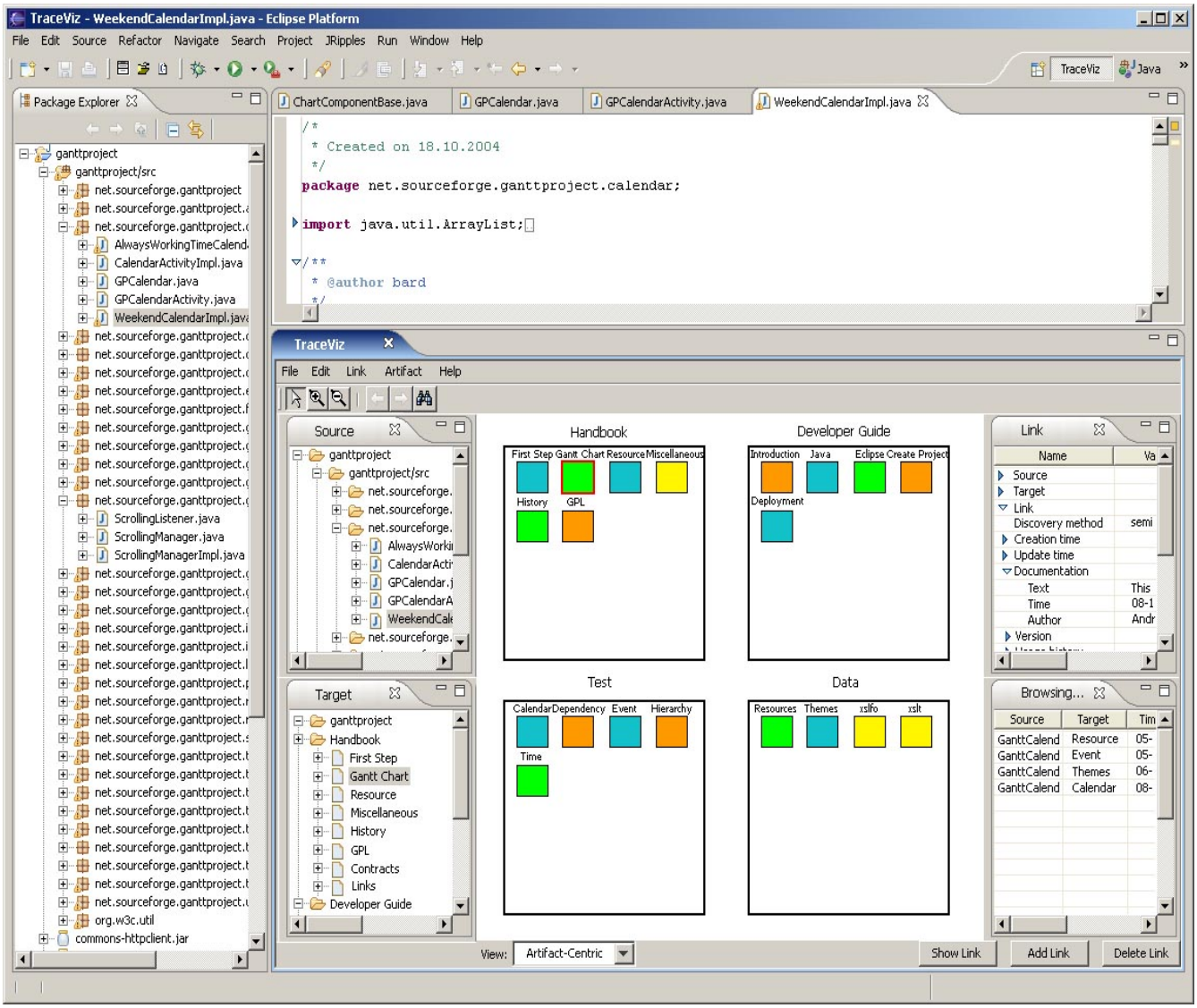

Figure 2. The TraceViz interface. The TraceViz view has three main areas: (1) the *elements area*, on the left, which contains the source and target browsers; (2) the *link area*, in the middle, which shows the links for a specific source or target (one of them is selected in the view); (3) the *information area*, on the right, which contains the link properties and browsing history

view). The links are grouped into categories, based on the chosen view (at the bottom). Each small colored square corresponds to a link, while a group of links in a large, labeled square correspond to a type of links. Links colored with the same color also form another type of link, based on different attributes.

3. The *information area*, on the right, which contains the link properties and browsing history. The link panel shows all the available attributes of the link and their values, while the browsing history panel shows the sequence of the links and artifact that were visited by the user using TraceViz.

Additional components may be added to these as well.

5.2. Using TraceViz

Figure 2 shows how TraceViz visualizes traceability links for the GanttProject (http://sourceforge.net/projects/ganttproject/). This is an open-source Java software with external documentation available in the form of user and developer manuals. We manually recovered and encoded in XML the links for this example.

The Eclipse user, while editing any file in the project, may launch TraceViz and the edited document becomes automatically the source of the traceability links. This is highlighted in the source browser. In Figure 2 the WeekendCalendarImpl.java file is the selected source for the traceability links. All links that have this file as their source artifact are represented as small squares in the middle, link area.

The links are grouped in larger squares, corresponding to the artifact-centric view (selected at the bottom of the window). This GanttProject file is linked to four types of documents from: *Handbook*, *Developer Guide*, *Test*, and *Data*. Each group of documents may be browsed directly from the link area or the target area.

In addition, color is used to denote the classification proposed in [5] (see section 2 and Table 1)

Table 1. Mapping of the types of links defined in [5] to colors. The user can define specific mapping.

	Normal links
	Lost Links
	Warning Links
	False Positives

Every link in the link area may be selected by a simple mouse click. Upon selection, the corresponding square is highlighted – in Figure 2 the second link from the top left is selected. This link has the WeekendCalendarImpl.java source, the GanttChart target, which is in the Handbook and it was marked as false positive (green) during recovery. Once the link is selected, the *Link* panel on the right displays all the attributes of the selected link (see Figure 2). Also, the Browsing history (on the right hand side) will capture the selection of the link and the time. Selected links may be deleted or edited – any information from the four browsers may be changed.

There is a third way to access links in TraceViz, through the target browser. By selecting any document from there, TraceViz will display all links that have that document as a target element. The user can choose to refresh the screen automatically or not. If the screen is refreshed, the previously displayed links are deleted, if not they are rearranged with the new ones. So, one can visualize for example all links that have WeekendCalendarImpl.java as the source artifact together with all links that have GanttChart as the target artifact.

For maintaining traceability links TraceViz supports the following tasks: defining link type, mapping links types to colors, adding/deleting/modifying traceability links, and documenting traceability links. The user can create documentation for the links with a predefined format, which can be linked to the corresponding data in the XML file.

The source and target browsers may also be used to add new links with the buttons at the bottom of the screen. Once a link is added, the user is required to insert all the elements and attributes of the link.

The link attribute browser (top right) may be used not only for editing the link attributes, but also to define views and categories. This feature is yet to be implemented.

To better support browsing, TraceViz allows query-based artifact/link search, filtering and sorting traceability links and performing logical operations on sets of artifacts and links. TraceViz also provides keyword searching; the user needs to input the keyword and choose the category for searching such as artifact name, link creation time, etc. Regular expression matching is used to find the results. The results are returned in the link area. The displayed links may be filtered; the user needs to choose the category first and define the condition for the filter such as link update time between 11:00 05/14/2004 and 22:00 08/20/2005. A sorting function allows the user to re-order the display based on values of a specified attribute – by default is the source name.

6. RELATED WORK

Though no existing work specifically addresses the opportunity of visualizing traceability links, there are several tools that deal with traceability links and have a visualization component.

INCOSE (International Council on System Engineering) [9] published a list of requirements for traceability tools within a Software Engineering Taxonomy. Some of the tools in this list implement several of the requirements that we formulated earlier. A few provide support for recording, displaying, and checking the completeness of installed traces: DOORS [21], TOOR [15], Rational RequisitePro [19], RDD-100 [8], etc. The current traceability environments also allow the definition of project-specific traceability data types. Some tools permit user defining new data types by either copying a predefined data type (e.g., CORE [23], icCONCEPT RTM [3]), or sub-typing a predefined data type (e.g., SLATE [20]), or creating an instance of a generic meta data type (e.g., RDD-100).

RDD-100 uses a basic entity-relationship structure to distinguish nodes and links, and allow the user to introduce distinctions between different types of nodes and links.

SLATE, which integrates requirements and design models, includes a requirements management tool, a scheduling engine that handles events and actions, and a simulation tool that utilized the TCL scripting language and interfaces with a wide variety of simulation models.

More details on these tools are presented in [6].

7. CONCLUSIONS AND FUTURE WORK

We argued in this paper for the need of visualizing traceability links to support the users in recovering, browsing, and maintaining them. Some situations lend themselves better to the use of visualization than others. In particular, when the developers already use analysis and comprehension tools that have a visualization component, when simultaneous display of links from multiple sources is desired, and when we need to see the attributes of the traceability links, in addition to their source and target artifacts.

In consequence, we formulated a set of high level requirements for visualizing traceability links. We plan to augment these requirements based on discussions within the research community.

The prototype tool we presented is based on these requirements and implements some of them. Future work will see the tool evolving to address all requirements.

8. ACKNOWLEDGEMENTS

This research was supported in part by a grant from the National Science Foundation (CCF-0438970).

9. REFERENCES

[1] Antoniol, G., Canfora, G., Casazza, G., and De Lucia, A., "Maintaining Traceability Links During Object-Oriented Software Evolution", Software - Practice and Experience, vol. 31, no. 4, April 2001, pp. 331-355.

[2] Antoniol, G., Canfora, G., Casazza, G., De Lucia, A., and Merlo, E., "Recovering Traceability Links between Code and Documentation", IEEE Transactions on Software Engineering, vol. 28, no. 10, October 2002, pp. 970 - 983.

[3] Chipware, "icCONCEPT tool - Replace RTM", Date Accessed: August, http://www.chipware.com, 2005.

[4] Cleland-Huang, J., Chang, C. K., and Christensen, M., "Event-Based Traceability for Managing Evolutionary Change", IEEE Transactions on Software Engineering, vol. 29, no. 9, 2003, pp. 796-810.

[5] De Lucia, A., Fasano, F., Oliveto, R., and Tortora, G., "ADAMS Re-Trace: a Traceability Recovery Tool", in Proceedings of 9th European Conference on Software Maintenance and Reengineering (CSMR'05), 2005.

[6] Dömges, R. and Pohl, K., "Adapting Traceability Environments to Project-Specific Needs", Communications of the ACM, vol. 41, no. 12, 1998, pp. 54-62.

[7] Hayes, J. H., Dekhtyar, A., and Osborne, J., "Improving requirements tracing via information retrieval", in Proceedings of 11th IEEE International Conference on Requirements Engineering, September 2003, pp. 138-147.

[8] Holagent, "Holagent Corporation product RDD-100", Date Accessed: August, http://www.holagent.com/products/product1.html, 2005.

[9] Incose, "SE Tools Taxonomy - Requirements Traceability Tools", Date Accessed: August, http://www.incose.org/productspubs/products/setools/tooltax/reqtrace_tools.html, 2005.

[10] Knethen, A., "Automatic change support based on a trace model", in Proceedings of 17th Conference On Automated Software Engineering, Edinburgh, Scotland, 2002.

[11] Lintern, R., Michaud, J., Storey, M.-A., and Wu, X., "Plugging-in visualization: experiences integrating a visualization tool with Eclipse", in Proceedings of ACM Symposium on Software Visualization, 2003, pp. 47-57.

[12] Maletic, J. I., Munson, E., Marcus, A., and Nguyen, T., "Combining Traceability Link Recovery with Conformance Analysis via a Formal Hypertext Model", in Proceedings of 2nd International Workshop on Traceability in Emerging Forms of Software Engineering, Montreal, Canada, October 7th, 2003 2003, pp. 47-54.

[13] Marcus, A., Maletic, J. I., and Sergeyev, A., "Recovery of Traceability Links Between Software Documentation and Source Code", nternational Journal of Software Engineering and Knowledge Engineering, vol. 15, no. 4, October 2005, pp. to appear.

[14] Pfleeger, S. L. and Bohner, S. A., "A Framework for Software Maintenance Metrics", in Proceedings of Conference on Software Maintenance, 1990, pp. 320-327.

[15] Pinheiro, F. and Goguen, J., "An Object-Oriented Tool for Tracing Requirements", IEEE Software 1996.

[16] Pohl, K., "PRO-ART: Enabling Requirements Pre-Traceability", in Proceedings of 2nd International Conference on Requirement Engineering, 1996, pp. 76-84.

[17] Ramesh, B. and Jarke, M., "Toward Reference Models for Requirements Traceability", IEEE Transactions on Software Engineering, vol. 27, no. 1, 2001, pp. 58-93.

[18] Ramesh, B., Tiwana, A., and Mohan, K., "Supporting Information Product and Service Families with Traceability", in Proceedings of 4th Workshop on Product Family Engineering (PFE-4), 2002, pp. 353-363.

[19] Rational, "Rational RequisitePro web site", Date Accessed: August, http://www.rational.com/products/reqpro/index.jsp, 2005.

[20] Tdtech, "System Level Automation Tool for Engineers (SLATE)", Date Accessed: August, http://www.tdtech.com, 2005.

[21] Telelogic, "Telelogic product DOORS", Date Accessed: August, http://www.telelogic.com, 2005.

[22] Toranzo, M. and Castro, J., "A comprehensive traceability model to support the design of interactive systems", in Proceedings of ECOOP Workshops, 1999, pp. 283-284.

[23] Vitech, "Vitech Corporation product CORE", Date Accessed: August, http://www.vitechcorp.com, 2005.