URC97034

An Approach to Building a Traceability Tool for Software Development

Nelly Delgado and 'Tom Watson
Department of Computer Science
The University of Texas at El Paso
El Paso, TX 79968
ndelgado, twatson@cs.utep.edu

1 Introduction

It is difficult in a large, complex computer program to ensure that it meets the specified requirements. As the program evolves over time, all program constraints originally elicited during the requirements phase must be maintained. in addition, during the life cycle of the program, requirements typically change and the program must consistently reflect those changes. Imagine the following scenario. Company X wants to develop a system to automate its assembly line. With such a large system, there are many different stakeholders, e.g., managers, experts such as industrial and mechanical engineers, and end-users. Requirements would be elicited from all of the stakeholders involved in the system with each stakeholder contributing their point of view to the requirements. For example, some of the requirements provided by an industrial engineer may concern the movement of parts through the assembly line. A point of view provided by the electrical engineer may be reflected in constraints concerning maximum power usage. End-users may be concerned with comfort and safety issues, whereas managers are concerned with the efficiency of the operation. With so many points of view affecting the requirements, it, is difficult to manage them, communicate information to relevant stakeholders, and it is likely that conflicts in the requirements will arise. In the coding process, the implementors will make additional assumptions and interpretations on the design and the requirements of tile system. During any stage of development, stakeholders may request that a requirement be added or changed. in such a dynamic environment, it is difficult to guarantee that the system will preserve the current set, of requirements.

Tracing, the mapping between objects in the artifacts of the system being developed, addresses this issue. Artifacts encompass documents such as the system definition, interview transcripts, memoranda, the software requirements specification, user's manuals, the functional specifications, design reports, and system code. Tracing helps 1) validate system features against, the requirement specification, 2) identify error sources and, most importantly, 3) manage change [4]. With so many people involved in the development of the system, it becomes necessary to identify the reasons behind the design requirements or the implementation decisions.

'This paper is concerned with an approach that maps documents to constraints that capture properties of and relationships between the objects being modeled by the program. Section 2 provides the reader with a background on traceability tools. Section 3 gives a brief description of the context monitoring system on which the approach suggested in this paper is based. Section 4 presents an overview of our approach to providing traceability. The last section presents our future direction of research.

2 Background

The typical approach to maintaining traceability, especially for complex systems, requires that all system artifacts created at various stages of the development process be linked to the requirements [5]. In such an approach, there must be hyperlinks (physical links) between all artifacts and requirements. These links should provide bidirectional, vertical and horizontal traceability. Bidirectional *traceability* refers to the ability to trace both backward and forward. *Vertical traceability* allows the user to trace between documents developed from different life cycles, whereas, *horizontal traceability* refers to the links between related objects created in the same life cycle [5]. Each document must have a logical structure so that the tracing tools will understand the interrelationships between different software documents and have the ability to update the links as the

system evolves [6]. Wit h a large volume of documents, however, it is difficult to maintain and update the links between the artifacts.

Through the links, tracing can also provide information concerning accountability for requirements, design, and implementation decisions, The ability to track projects and manage design rationale are other uses for traceability. The rest of this section deals with the traditional approach to tracing, i.e., tracking the requirements to the implementation.

Some approaches to building a traceability tool include: an object-oriented approach, a graph-based approach, and an approach that involves the management through a project database. In the object-oriented approach, users define the classes of artifacts and the relationships between them. The classes of artifacts provide the logical structure necessary for the documents to be traced. The relationship classes provide the structure for defining links and their relations. The use of relations instead of simple links lets developers distinguish among different links between the same objects. Also, by using properties of relations, this approach can relate objects that are not directly linked [4].

The graph-based approach takes both the coarse-grain level and the fine-grain level of the system into account. On the coarse-grain level, links represent dependencies between whole documents. On the fine-grain level, the structure of the documents are taken into account. Both levels are necessary to provide for adequate traceability. The collection of document.s is represented by the use of hierarchical graphs. Operations on the graphs are defined by means of a formal language based on a graph rewriting system [6], On the coarse-grain level, ail project documents and the relations between the documents are represented in the graph, On the fine-grain level, links represent the relation between individual objects in documents. These objects may be contained in the same document or in separate documents.

The project database model involves a database management system and an object-based model of software life cycles. All of the documents created during the development of the system are stored in a project database [3]. Similar to previous approaches, this approach requires that documents and relations are highly structured. Using a predefine document structure and a set of document relationships, documents are developed to allow the management of links. Through the use of a database, stored documents can be written in either natural or formal languages. Key words and key elements need to be identified as the user creates the documents to provide points for tracing across documents [3].

These approaches to developing traceability tools all have one thing in common-they link all artifacts to requirements. The advantage to having direct links' between all artifacts is that one can trace the system and ensure that the system requirements have been met, thus providing more complete coverage of the system. In a system with thousands of interrelated objects, however, getting all of the information available may not, be useful. Too much information could provide the user with an over abundance of irrelevant data [4].

3 An Overview of the Context Monitoring System

Context monitoring is an approach for managing properties and relationships on data of a program. Context monitoring consists of the following:

- the specification of integrity constraints on data or objects being modeled by a software system, and
- a constraint satisfiability mechanism that verifies their enforcement during the program's execution.

Integrity constraints are the conditions that data maintained in a knowledge base must satisfy as it evolves. Specified in a typed first-order logic language [2], they formalize properties about data objects and the program.

Fig. 1 provides an overview of the context, monitoring system. The constraints capture properties, relationships, restrictions and limitations on data or objects of the program and may be used to reflect the interpretations and assumptions that, are made about the objects during development. The constraints are elicited from numerous sources, denoted by S_i in the figure, that includes the customer, domain experts, analysts, and members of the design, implementation and maintenance team.

The constraints reside in a repository with links to the documentation. This may be in the form of a data dictionary, requirements definition, software requirements specification, design document, user's manual, program documentation, memoranda, interview transcripts, or videotape transcripts. The documents may reside in a project database or maintained as separate text files. If a document is not kept on-line, then the linked file would contain merely the location of the document and the location of the constraint in that document,

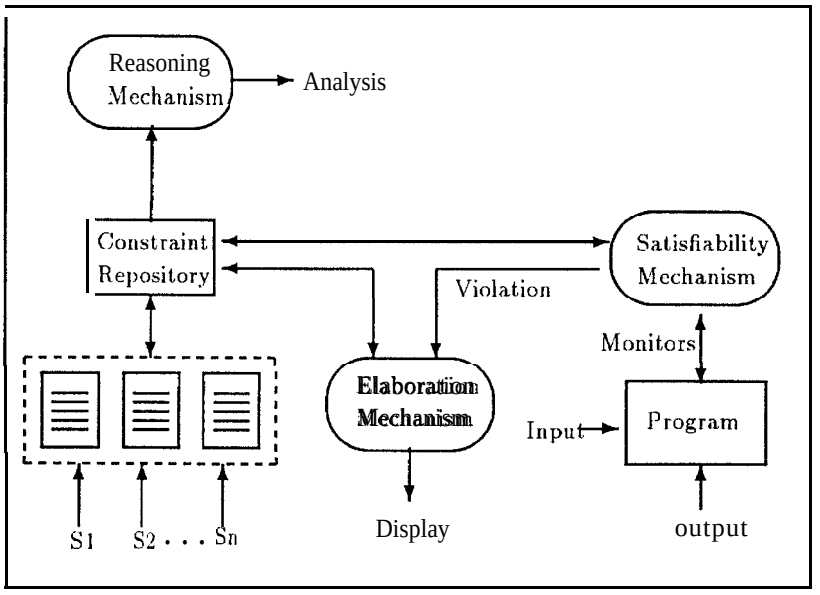

Figure 1: Overview of the context monitoring system.

The constraint satisfiability mechanism provides dynamic verification that, the actual behavior of the program corresponds to the intended behavior of the program. When a constraint violation occurs, the monitoring mechanism captures the state of the machine, both prior to the violation and at the time of the violation. A violation may be an indication of one of the following:

- The program does not maintain the constraint.
- The input data does not meet specified properties.
- Conflicting constraints exist, which cause at least one of the constraints to be violated.
- A change to the program has violated an existing constraint,
- The environmental context in which the program is running has changed, invalidating the assumption made while designing and coding the program.

The user is supplied with information concerning the Violation through the elaboration mechanism. Because the documents are linked to the constraints, the user has access to the source(s) of the constraint, justifications, and other information that, may be contained in the documentation. The reasoning mechanism is a tool that groups related constraints for analysis and determines potential inconsistencies between constraints apart from the program itself. This permits static analysis of the knowledge collected from multiple sources before too much time has been invested in the design and/or implementation.

It is important to note that with this approach, the constraints are not embedded in the program, but are maintained independent of the program. Separating the integrity constraints ensures that changes in the code do not inadvertently change a constraint and, through constraint satisfiability, that added or changed code does not violate existing constraints.

4 Traceability Using Context Monitoring

The approach suggested here provides a way for linking between requirements, constraints, and parts of the program that, manipulate constrained variables. A distinguishing characteristic is that the developer does not have to create physical links between requirements and the implementation. 'This simplifies tracing because constraints are not embedded in the program and, as a result, the user does not have to worry about changes in the code that may alter links. Tracing in the context monitoring system only involves physics!

links between the constraints stored in the repository and the documents as shown in Fig. 2. Although this approach does not provide links between all requirements and program segments, it does provide a way to link a significant subset of the requirements.

Another characteristic of this traceability approach addresses fluctuating and conflicting requirements, a major issue in software development [I]. Consider a case where conflicting constraints are specified on a requirement. In such a case, the satisfiability mechanism will detect a violation if it is impossible for the program to satisfy both requirements. Through traceability, the user will have access to the sources of the constraint providing a basis for resolving the conflict. In addition, the traceability tool is not, preoccupied with managing constraints. Because the constraints are not embedded in the program, even if implementation completely changes, the links between constraints and the documents remain the same unless the constraints are deleted.

The approach presented in this paper provides the means to physically link the constraints to the various requirement documents through bidirectional hyperlinks. Because only those parts of the requirements and documents that specify a constraint are linked to the corresponding formal constraint and links are not maintained in the code, the number of links to be managed is reduced. In other words, constraints should always be tied directly to a document.

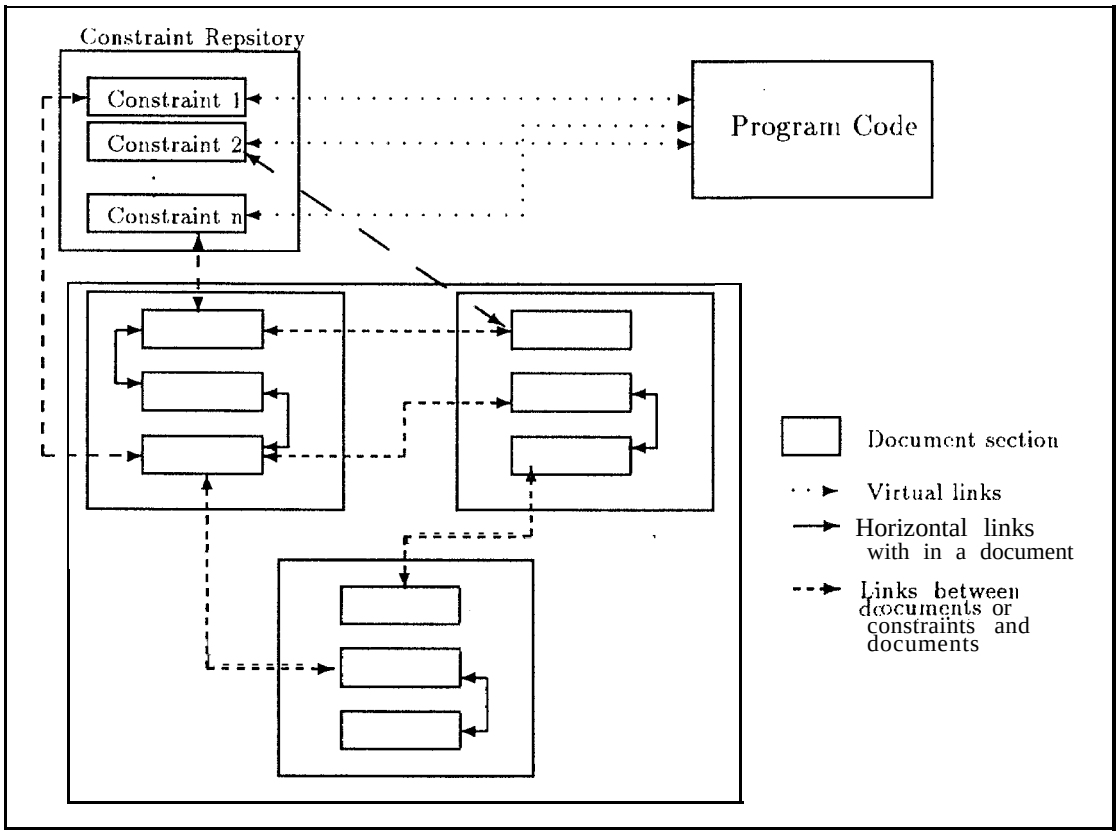

Figure 2: Tracing in the context monitoring environment.

The context monitoring system provides for bidirectional, vertical and horizontal traceability, and also allows the user to check for accountability, design rationale and dependencies. Project tracking, however, cannot be done with this approach. Two situations may occur. One is that a violation occurs during program execution and the user wants to determine the source of the constraint. Through the elaboration mechanism, it is possible to return to the repository based on the constraint, violated; therefore, the user can link back to the document. A second situation is when the user requests information concerning the impact of a requirement, or a constraint on a program. In this case, the satisfiability mechanism gives information on the code that is affected by this constraint, providing a virtual link between the constraint. and the code. Similarly, this procedure can be followed to go from the code to the constraint repository. Clearly, the links

between the constraints and the documents are fine-grain and the virtual links between the constraints and the program code are coarse-grain, (making the links between the code and any documents coarse-grain as well). As a result, any queries on the coarse-grain links will lead the user to candidate areas of interest in either the code or the documents. Recall that the relationship between these documents is based on the constrained variables.

Related constraints from different documents are linked together providing both horizontal and vertical traceability. This allows the user to determine if a constraint is supported by more than one document or decision. These links may be contained in the same document or across multiple documents providing horizontal traceability with a tine-grain level.

Because all constraints in the system can be linked back to a decision in the documentation through vertical links, design rationale can be easily traced. Provided that the documents contain the information on whom originally specified the requirement, the resulting constraint can be linked to this requirement, providing accountability to the decision makers.

5 Summary

The traceability tool in the context monitoring system allows virtual links from the code to the constraints providing tracing between requirements that specify relationships between properties of objects on the system. The advantage to this approach is that the system does not, have to provide hard links directly to the code; therefore, when modifications are made in the code, links are not inadvertently affected. Additionally, this allows the user to trace only the relevant information corresponding to a constraint. One disadvantage to this approach is that it, is not possible to ensure that all system requirements are met because the constraints capture only those parts of the requirements that specify properties and relationships between objects.

Future research in the area of traceability tools in context monitoring includes many different goals. First, we must determine the best method for storing the documents of the system. As shown in the discussion of traceability tools in section 2, this must be a highly structured approach to allow the system to process the document links. Some possible approaches to providing this structure include the ot)jcct-oriented, the graph-based and the project database management approach.

The next objective is to develop a prototype to model the proposed traceability tool. This prototype will provide the tracing technique to the context monitoring system. The elaboration mechanism, constraint repository and documents will be included in the implementation of this prototype.

Acknowledgments. This work was supported by NASA under contracts NAG-1012 and NCCW-0089, and NSF grant no. CDA-9522207.

References

[1] Curtis, B., H. Krasner and N. Iscoe, "A Field Study of the Software Design Process for Large Systems", *communications of the ACM, 31* (11), pp. 1268-1287. Nov. 1988.

[2] Gates, A. and F. Fernandez, "Building Systems with Integrity Constraints," to be published in *The Proceedings of the Second World Conference on Integrated Design and Process Technology,* Dec. 1-4, 1996, Austin, Texas.

[3] Horowitz, E. and R. Williamson, "SODOS: A Software Documentation Support Environment–Its Use", *IEEE Transactions on Software Engineering,* SE-**12**(11), pp. 1076-1087, Nov. 1986.

[4] Pinheiro, F. and J. Goguen, "An Object-Oriented Tool for Tracing Requirements", *IEEE Software*, pp. 52-64, Mar. 1996.

[5] Ramesh, B. and M. Edwards, "Issues in the Development of a Requirements Traceability Model", *Proceedings of the IEEE International Symposium on Requirements Engineering,* San Diego, CA: IEEE Computer Society Press, 1993, pp. 256-259.

[6] Westfechtel, B., "A Graph-Based Approach to the Construction of Tools for the Life Cycle", *IEEE*, pp. 2-13, June 1992.