NFR-Assistant: Tool Support for Achieving Quality

Quan Tran and Lawrence Chung
School of Computer Science
University of Texas at Dallas
2601N. Floyd Rd., Richardson, TX 75083
(972) 883-2185
{ttsx19c, chung}@utdallas.edu

Abstract

This paper presents the NFR-Assistant, a prototype CASE tool, which assists the software developer in systematically achieving quality requirements. The tool allows for explicit representation of non-functional requirements, consideration of design alternatives, analysis of design trade-offs, rationalization of a design choice and evaluation of the level of achievement of NFRs. As one of the first tools, the particular prototype presented in the current paper is a Java applet rendition of a subset of the NFR-Assistant. Interestingly this paper illustrates the use of the prototype for the development of an architectural design for no other than a distributed version of the NFR-Assistant itself.

1 Introduction

Achieving such requirements as evolvability, performance, and distributedness is often crucial to the success of a software system. Despite the increasing concerns for such quality requirements (*non-functional requirements* or *NFRs*), the software engineering community has placed a lop-sided emphasis on functional requirements in the past. This has resulted in the development of a rich set of notations and methodologies for systematically addressing functional requirements and supporting the process of generating software that meets the functional requirements. This lop-sided emphasis, however, has also resulted in our inability to explicitly represent quality requirements, and to generate software in such a way that the quality requirements can be closely related to the components of the software, i.e., the way functional requirements are. One tactic to alleviate this difficulty is to address non-functional requirements throughout the entire software engineering cycle. This approach leads to two distinct shifts of emphasis. First, the consideration of non-functional requirements brings about a more goal-oriented perspective in that the objectives and intent of the customers have more influence upon the design of the solution. Second, the aim is shifted to more of a process-oriented approach in that these goals are continually reinforced throughout the engineering process and not just measured in the finished product. Consequently, all of the quality and resource issues may be considered and resolved in due process. Thus, this emphasis effectively strives to build quality into the product, and thereby, attempts to minimize rework.

2 NFR Framework

The Non-Functional Requirements framework provides a simple ontology to methodically address resource constraints such as cost, time schedule, and personnel; and quality attributes desired by the customer such as reliability, performance, and user-friendliness. Its language offers expressive constructs for documenting NFRs, and its qualitative, logical reasoning offers an intuitive system to explore these soft issues.

To handle the complexity of numerous NFRs of any large-scale software product, the framework uses a systematic approach to NFR analysis and design, and software trace ability. The NFR framework establishes a network of *softgoals* connected by *interdependency links*, and represents this graphically to alleviate documentation difficulties. It supports an evaluation procedure to trace the effects of the softgoals and interdependency links. Moreover, the framework leverages accepted standards and knowledge gained from previous requirements analysis by supplying a *method catalog* of softgoal decompositions to clarify softgoals and a *correlation catalog* of interdependency links to formulate implicit relationships.

2.1 Softgoal

The NFR framework endorses a goal-oriented approach. The notion of a goal is refined into three separate intents of *softgoals*, *operationalizing goals*, and *claim softgoals*. The term *softgoal* import a finer connotation than non-functional requirements in that these "soft" goals are the lofty customer wants (e.g., user-friendliness) expressed in an imprecise manner. Next, o*perationalizing goal*s comprise of possible solutions or design alternatives to achieve the softgoal. For example, a graphical user interface (GUI) can be offered as an operationalized goal in lieu of the desired softgoal of user-friendliness. Lastly, *claim softgoals* argue the rationale and explain the context for a softgoal or interdependency link. Continuing with the small example of user-friendliness, a claim softgoal can argue that GUIs help “visual juxtaposition” when the graphical layout groups relevant information concisely.

2.2 Interdependency Links

Interdependency links represent relationships between softgoals and/or other interdependency links. These relationships are categorized within an intuitive, qualitative range. Listed in an increasingly positive magnitude, these interactions include BREAK, HURT, NUKN, HELP, and MAKE. As their names suggest, BREAK creates a sure negative effect, HURT imposes a partial negative effect. HELP triggers a partially positive effect, and MAKE ensures a totally positive effect. (More details can be found in [2].)

Interdependency links can also be combined among multiple softgoals and/or interdependency links through AND or OR composition. AND describes the relationship in which all the refined subgoals must be met for the goal to be met. OR defines the relationship that at least one of the refined subgoals must be met for the goal to met.

2.1 Evaluation

The evaluation procedure determines the degree, via labeling, to which any given non-functional requirement is being addressed by a set of design decisions. The label of a softgoal or contribution reflects its status of being *satisfied*, *denied*, *conflicting*, or *undetermined*. Satisfied means being satisficeable, practically satisfying a majority of customers, and not deniable. Denied means to be deniable, but not satisficeable. Conflicting means being both satisficeable and deniable. Undetermined means being neither satisficeable or deniable.

The evaluation procedure consists of a two step process. First, the individual effects between two softgoals and/or interdependency links are calculated such that the source is the cause that would produce an effect upon its destination given their interdependency link. For example, a source that is satisficed would BREAK its destination by denying it. Next, the AND and OR relationships are merged. Finally, all of the attributing effects are combined to produce an overall effect [2].

3 NFR-Assistant CASE tool

As a computer-aided software engineering (CASE) tool, the NFR-Assistant incorporates the semantics and syntax of the NFR framework into a practical application to aid software developers in analyzing non-functional requirements. It operates the error-prone task of incrementally tracing the effects of numerous NFRs and still endows the developer with full control over the decision-making process by providing a semi-automated, interactive evaluation procedure. The NFR-Assistant is one of the first tools that allow for a (semi-formal) representation and semi-automatic, systematic development process, with the additional features of design rationale.

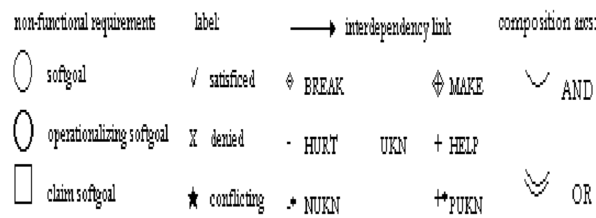

Figure 1: NFR-Assistant Notation

3.1 Related Work

The services offered by the NFR-Assistant are similar in spirit to those of SYBIL [9], which extends earlier work on decision support systems, in particular gIBIS [4]. SYBIL manages goal relationships and alternative decisions, promotes reuse of previous design decisions, and evaluates goal satisfaction. Unlike SYBIL, however, the main focus of the NFR-Assistant lies in NFRs. The NFR-Assistant is also similar to recent work by Boehm and In that explores a knowledge-based tool for identifying potential conflicts among NFRs early in the software / system life cycle [1]. The NFR-Assistant grows out of the earlier Mapping-Assistant of the DAIDA environment that provides support for all phases of information system engineering [7]. The Mapping-Assistant [2] aids the developer by offering dependency types, which suggest basic alternatives for the mapping of functional requirements objects into design objects, and then into implementation objects. The NFR-Assistant complements and extends the Mapping-Assistant by allowing for the representation of NFRS as softgoals in terms of which design decisions are justified. In a similar vein, KATE takes a goal-oriented approach, here functional requirements specification objects as goals, and offers services for transforming an incomplete and inconsistent requirements specification into a fuller one, which is later transformed into an implementation [5]. Quality Function Deployment (or the House of Quality) [6] in industrial engineering has been applied to Software Quality Assurance (e.g., [10]). QFD and the NFR-Framework are similar in that both can be used as media for communication and planning and to provide a conceptual map from customers' requirements to designs and implementations.

The Java rendition of a subset of the NFR-Assistant [14] exploits those concepts that were developed earlier for a "proof-of-concept" implementation of the NFR Framework in Prolog using the client-server architecture of ConceptBase [8]. This Java rendition can be viewed as a cousin of OME, a tool which is being implemented in Java and a knowledge base management system [11] to provide support for agent-oriented requirements analysis [15].

4 Reflective Case Study

Since the NFR-Assistant CASE tool strives to facilitate the decision-making process in any software development. An interesting challenge would be to carry out a meta-analysis of the NFR-Assistant itself, since it is after all a software system in its own right. With this interesting prospect, this study resolved to utilize a tacit conceptual prototype of the NFR-Assistant in developing the CASE tool. Once created, the tool then re-enacted the development process as to verify its use in defining the customers' goals, analyzing the available solution alternatives, and finally rationalizing the selected solution. The formalized study follows in the next section.

In this simplified analysis, the causes and effects were easily traceable, and a superior system architecture was determined that clearly surpassed the other options as it fulfilled a majority of the quality requirements. In such straight-forward decisions like this where only a tolerable amount of factors were considered, the full power of the CASE tool might not be realized. However, scale this explanatory study to that of an unwieldy legacy system with numerous factors and consequences, and one would gather an appreciation for the assistance of such a tool. Moreover, the automated documentation by-product will be greatly instrumental in keeping re-engineering efforts consistent to early decisions as extensions and additions are developed upon the existing legacy system.

4.1 Goal-oriented approach

To derive the non-functional requirements for the NFR-Assistant, consider its general usage scenario. This collaborative process may iterate in any order or occur simultaneously, and all information is to be analyzed and recorded via the NFR-Assistant.

> *"An information engineer plots several abstract goals and desirable components given by the end customers along with corresponding claims /rationale. A software developer offers solution alternatives and establishes the resulting effects. The project manager imposes constraints and resources."*

From the above usage scenario and through consultation with the stakeholders, the most critical underlying non-functional requirement for the NFR-Assistant proves to be the ability to support a collaborative groupware environment where all its users can use the CASE tool in a joint effort. In order to produce this collaborative groupware, a distributed network must be established. Leveraging relationships found within the method catalog, an available, intact, and maintainable communication medium proves to be reliable. Platform independence will extend the network's availability. Locally, each individual NFR-Assistant interface must be interactive to handle the changing requests from its multiple users, and the accruing information must be consistently presented. In addition, user-friendliness for the human-computer interface (HCI) is highly desirable so as to accommodate such a diverse profile of end-users. Lastly, being a prototype software, extendibility and evolvability are paramount to the life of the NFR-Assistant. The set of non-functional requirements considered and their synergies are displayed in Figure 2. With these end goals in mind, the system architecture can then be decided upon rationally and systematically.

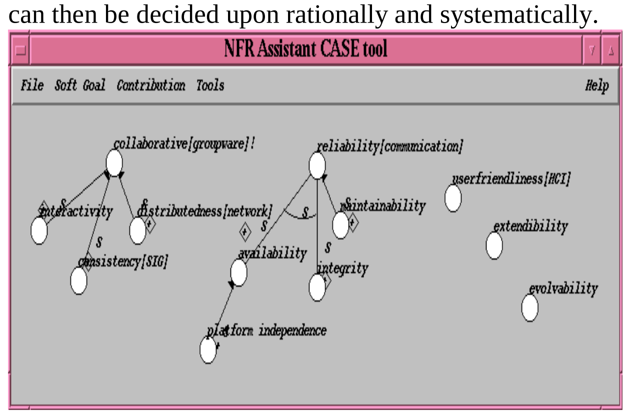

Figure 2: NFRs for the NFR-Assistant CASE tool

4.2 System Architecture Analysis

Before one particular system architecture can be selected, an understanding of the tool's fundamental architectural components and control flow was defined as follows. From some medium, the appropriate input given from user transactions are collected and prepared. This data is processed within the context of the NFR framework to create a softgoal interdependency graph. Method and correlation catalogs may be leveraged in this step. The *SIG* is evaluated, and displayed to an output medium. These general constructs of input, SIG, catalog, evaluation, display, and output are similarly denoted throughout the following discussion of architecture analysis and selection. Also, Figure 7 depicts the life of the SIG generated by the NFR-Assistant during this architectural analysis.

A few software architecture models including those of the *Pipe-and-Filter*, *Implicit Invocation*, *Abstract Data Type* and *Client/Server* paradigms were proposed for the NFR-Assistant CASE tool and molded the general system architecture with their distinctive characteristics. Each resulting software architecture scheme was analyzed with respect to the prescribed set of non-functional requirements of the NFR-Assistant CASE tool to evaluate its advantages and disadvantages. Thus, with each clearly defined, proposed architecture as an operationalizing goal and its relationships readily linked among the established non-functional requirements, the NFR-Assistant CASE tool is able to forecast the effects of each design alternative.

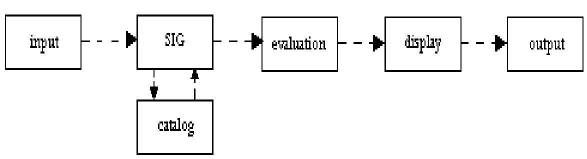

Figure 3: Pipe and Filter

The Pipe-and-Filter architecture, shown above, provides a very simple, direct process. It is generally most appropriate in applications where completely independent tasks are to be performed in succession. However, it is not amenable to random user input. Given the common usage scenario of this CASE tool to handle dynamic changes via user input, the Pipe-and-Filter architecture proves to be detrimental. Hence, it would hurt projections of interactivity. Consequently, this analysis can be captured by Figure 7 by establishing a "BREAK" interdependency link between the operationalized goal " pipe-and-filter" and the softgoal interactivity". If this architecture was chosen, the NFR-Assistant would label the operationalized goal "pipe-and-filter" as satisficed, and proceed to propagate this information throughout the SIG thereby labeling the softgoal "interactivity" and its parent softgoal "collaborative[groupware]" as denied.

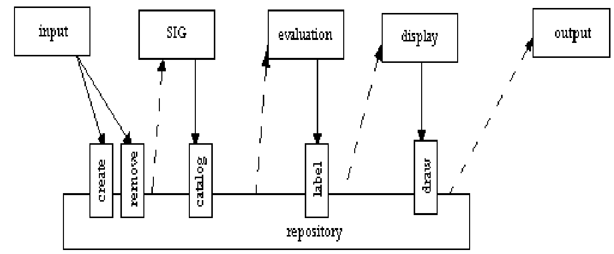

Figure 4: Implicit Invocation

The Implicit Invocation architectural style, shown above, is based on events and implicit invocations. A component/module announces (or broadcasts) one or more events; other components register an interest in an event by associating a procedure with the event. When the event is announced, the event demon (depicted as the *repository* module) invokes all the procedures that have been registered for the event. Therefore, the interactions/ connections between the components/ elements are not direct, but rather are constrained to implicit connections via the event demon.

Hence, the Implicit Invocation architecture caters to user input very nicely with its event system, so it solves the weakness of the pipe-and-filter model. This is reflected in Figure 7 by establishing a "MAKE" interdependency link between the operationalized goal "implicit invocation" and the softgoal "interactiviy". Its shared data structures within the *repository* module reduce the computational overhead of inter-process communication, and reduce the memory overhead. However, its consistency may suffer when multiple users attempt to access the shared data as is anticipated by the usage scenario. Figure 7 depicts this relationship by adding a "HURT" interdependency link between this proposed architecture and the softgoal "consistency". In short, implicit invocation serves appropriately for single users, but may not offer proper security and/or performance measures to support collaborative usage of the NFR-Assistant.

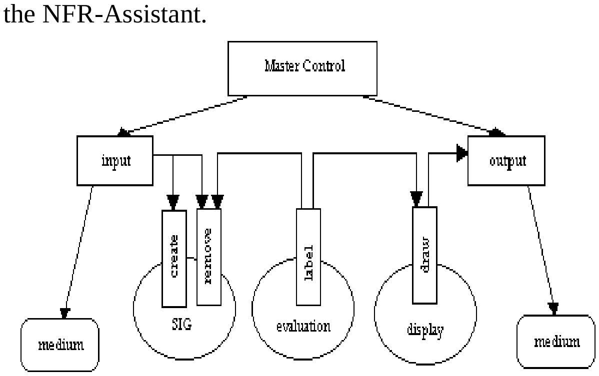

Figure 5: Abstract Data Type

The Abstract Data Type (ADT) architecture, shown above, uses data encapsulation, information hiding, and interface functions to ensure consistency during collaborative efforts of multiple users, thereby escaping the dilemma found within Implicit Invocation. Also, the object-oriented inheritance nurtures extendibility and evolvability. However, ADT does not support an event-driven system to readily respond to interactive user input.

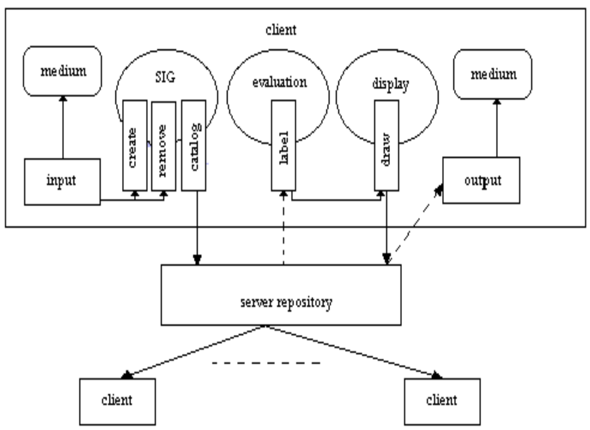

Figure 6: Web Client / Server ADT with Events

The *Web Client/Server Abstract Data Type with Events* architecture, shown above, is intended to amass all the benefits from the individual architectural styles to provide an appropriate and efficient model for the NFR-Assistant. As an ADT architecture, it inherits the data encapsulation and information hiding to ensure consistency of the NFR-Assistant CASE tool as well as object inheritance to achieve extendibility and evolvability. Also, its event-driven control flow facilitates interactivity. Moreover, its Client/Server architecture establishes a network foundation (i.e. the Internet) to handle distributed computing. The Web generates heavy processing for the client but very little processing for the server. When a web browser requests a Java applet, the Internet server initiates a separate TCP/IP session to download each applet within a Web page. The server retrieves the requested applet and sends the applet. The browser loads the applet into the client's memory and executes it, but deletes the applet from memory when the Web page is exited. The cooperative processing paradigm within the Client/Server architectural style is based on transparent sharing of resources. The components/elements are the number of computer distributed physically. These interact via connections through a communications network. A cooperative effort is achieved under the constraint that only a single set of resources be available to share among the components/elements. All these modules share the data structures providing knowledge about the graph although they communicate via protective ADT function interfaces. This client-sided application encapsulates each local NFR-Assistant CASE tool client. The server serves as an interface to other clients via the web network. The *server* will then encapsulate the shared data and feature the event-driven system.

Finally, Figure 7 shows the end results of the above analysis of potential software architecture candidates as they relate to the stated NFRs. The selected *Web Client/Server ADT with Events* architecture is labeled with a check mark, and its achieved quality requirements are similarly labeled with check marks. The history of the discarded architecture alternatives, labeled by cross marks, is archived by denying their interdependency links, thus voiding their contribution to the final network. Even still, the labeled interdependency links reflect the preceding discussion about how the architectures would have impacted the quality requirements. As a miscellaneous remark, it is interesting to note that the softgoals *platform independence* and *user-friendliness[HCI]* remain unaffected by any of the proposed architectures thus far.

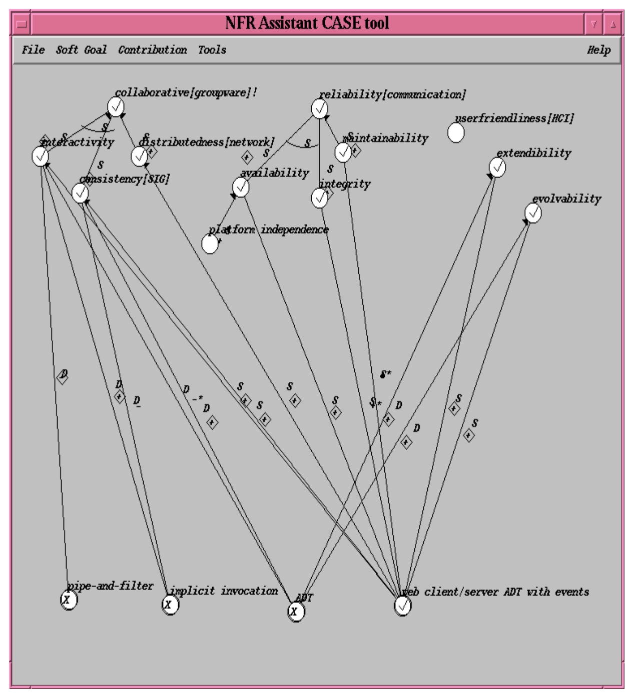

Figure 7: *SIG* of the NFR-Assistant's architecture decision process

5 Conclusion

The main technical contribution is a Java rendition of a subset of the NFR-Assistant [13], and the proposal of a software architecture for distributed processing of quality requirements. This Java applet NFR-Assistant

CASE tool serves as a practical prototype for the NFR framework, namely the construction of the SIG (Softgoal interdependency graph) and the incremental evaluation procedure. The NFR-Assistant readily handles revisions and changes to the SIG. The automatic evaluation relieves the software developer from this tedious, error-prone task. Its graphical user interface provides a very intuitive and friendly dialog between the CASE tool and software developer. These accomplishments will serve as a solid foundation in which to evolve the NFR-Assistant CASE tool parallel to the maturation of the NFR framework and the Java language.

There are several lines of future work. Collection of feedback and preliminary analysis is being reviewed for the working prototype that is presently available via the world-wide-web. Since the NFR-Assistant CASE tool is a visualization mechanism to represent and manipulate objects (i.e. NFRs), it is governed by the same limitations inherent in graph visualization. With increasing complexity, the graph representation becomes cluttered with intersecting edges and associated labels. The labeling issue may be addressed with alternative notations or visual elements such as color usage. The graph complexity may be lessened with planar graphs. As peccadilloes resulting from inaccurate coordinate display conversion, the arrowheads of single interdependency links and the arcs of combined interdependency links are graphically warped. Resourceful drawing techniques may remedy these nuances. Also, it would be interesting to examine whether the NFR-Assistant reduces rework and scrap, hence inducing shorter production time and lower cost, in the spirit of QFD.

Future work should also include implementation of methods and correlation rules so that knowledge of design techniques and tradeoffs can be captured and reused. The tool should be integrated with the Mapping Assistant that supports the representation and mapping of functional requirements. Also, this Java applet CASE tool may be mounted in a distributed and finally collaborative setting once a server is available. In addition, with the advancement of Java's file capabilities, this Java applet may be given memory capacity to be able to persist and load pertinent data for extended usage.

6 References

[1] B. Boehm and H. In, "Identifying Quality-Requirements Conflicts", *IEEE Software*, March 1996.

[2] L. Chung, B. A. Nixon, E. Yu and J. Mylopoulos, *Non-functional Requirements in Software Engineering*, Kluwer Publishing (to appear).

[3] L. Chung, D. Gross and E. Yu, "Architectural Design to Meet Stakeholder Requirements" *The First Working IFIP Conference on Software Architecture (WICSA1)*, 22-24 February 1999, San Antonio.

[4] J. Conklin and M. L. Begeman, "gIBIS: A Hypertext Tool for Explanatory Policy Discussions," *ACM Transactions on Office Information Systems*, 6(4), pp. 303-331, 1988.

[5] S. F. Fickas, Automating the Transformational Development of Software. IEEE TSE, Vol. SE-11, No. 11, November 1985, pp. 1268-1277.

[6] J. R. Hauser and D. Clausing, "The House of Quality," *Harvard Business Review*, May-June 1988, pp. 63-73.

[7] M. Jarke, J. Mylopoulos, J. W. Schmidt, Y. Vassiliou, "DAIDA: An Environment for Evolving Information Systems," *ACM Trans. Information Systems*, vol. 10, no. 1, January 1992, pp. 1-50.

[8] M. Jarke (Editor), ConceptBase V3.1 User Manual. Univ. of Passau, 1992.

[9] J. Lee, Extending the Potts and Bruns Model for Recording Design Rationale. *Proc. 13th Int. Conf. on Software Eng.*, Austin, Texas, May 13-17, 1991, pp. 114-125.

[10] T. J. McCabe and G. G. Schulmeyer, "The Pareto Principle Applied to Software Quality Assurance," In G. Gordon Schulmeyer and James I. McManus (Eds.) *Handbook of Software Quality Assurance*, New York: Van Nostrand Reinhold, 1987, pp. 178-210.

[11] J. Mylopoulos, A. Borgida, M. Jarke, and M. Koubarakis, Telos: Representing Knowledge about Information Systems, *ACM Transactions on Information Systems*, vol. 8, Oct. 1990, pp. 325-362.

[12] B. A. Nixon, *Dealing with Performance Requirements for Information Systems*. Ph.D. Thesis, Dept. of Computer Science, Univ. of Toronto, 1997.

[13] H. A. Simon, *The Sciences of the Artificial*, Second Edition. Cambridge, MA: The MIT Press, 1981.

[14] Q. Tran, "A CASE Tool for the Non-Functional Requirements Framework." M.S. Thesis, Dept. of Computer Science, Univ. of Texas Dallas, 1998.

[15] E. Yu, *Modelling Strategic Relationships for Process Reengineering*. Ph.D. Thesis, Dept. of Computer Science, Univ. of Toronto, 1995. Also Technical Report DKBS-TR-94-6.