A CASE TOOL FOR THE NON-FUNCTIONAL REQUIREMENTS FRAMEWORK

Quan T. Tran
ttsx19c@utdallas.edu

Lawrence Chung
chung@utdallas.edu

University of Texas at Dallas
P.O. Box 830688
Richardson, TX 75083-0688

ASSET '99
Call for Papers
October 15, 1998

ASSET paper abstract

Tool Support for Achieving Quality

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. 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. Exploiting Java's orientation towards the distributed processing paradigm, this rendition allows for distributed processing of quality requirements. 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.

1.0 Introduction

In industry, software engineering promotes the present, predominant paradigm of object-oriented analysis and design (OOAD). This strategy targets functional requirements, those customer guidelines stating what the software product is to perform. This product-oriented approach measures the degree to which a finished product meets non-functional requirements, those guidelines or requirements from the customer that describe how the product should perform. Thus, OOAD ensures that the delivered software product is functionally correct; however, it does not efficiently engineer the intangible non-functional requirements of quality standards and resource constraints. 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 strives to effectively build quality into the product, and thereby, attempts to minimize re-engineering.

A framework that incorporates the goal-oriented approach of non-functional requirements is presented in this paper. A brief overview of its basic constructs is explained conceptually and abstractly. Then, a CASE tool extraction of the framework is introduced. Its motivation is argued, and synopses of its implementation are displayed. An introspective case study is provided to reinforce the overview of the suggested ontology in addition to displaying the CASE

tool itself. Similar related works are noted, and a conclusion of this paper's contribution is presented.

2.0 NFR Framework

The Non-Functional Requirements (NFR) framework provides a simple ontology to methodically address non-functional requirements. Its qualitative, logical reasoning offers an intuitive system to explore these soft issues. Its language offers expressive constructs for documenting non-functional requirements.

Non-functional requirements include resource constraints such as cost, time schedule, and personnel; and quality attributes desired by the customer such as reliability, performance, and user-friendliness. Analogous to the business model of supply-and-demand, the inherent trade-off relationship between resource constraints and quality attributes of non-functional requirements offer an amendable exchange to compare advantages and disadvantages of software systems.

To handle the complexity of numerous non-functional requirements of any large-scale software product, a systematic approach must be able to catalog and trace the effects of these requirements clearly. The NFR framework strives to accomplish this very task. The framework provides a setting for non-functional requirements to be stated and denoted clearly, operational solutions to be evaluated, rationale to be claimed, direct and indirect relationships among non-functional requirements to be defined, and affected interdependency links to be traced. The NFR framework establishes a network of non-functional requirements in the form of *softgoals* connected by *interdependency links*, and represents this graphically to alleviate documentation. It supports an evaluation procedure to trace the effects of the softgoals and interdependency links. In addition, this framework attempts to leverage the experience of past software

development and accepted standards by supplying a *method catalog* of known softgoal decompositions to clarify softgoals and a *correlation catalog* of known interdependency links to formulate indirect relationships implicitly. Each of these components, with the exception of method and correlation catalogs, is discussed in some depth in the ensuing sections. The application case study used to explain the fundamental concepts of the NFR framework and to capture the functionality of the resulting NFR-Assistant CASE tool is no other than the NFR-Assistant CASE tool itself.

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 wants of the customer expressed in an imprecise manner. For example, user-friendliness, or reliability may be included as softgoals. *Operationalizing goal*s comprise of possible solutions or design alternatives to achieve the softgoal. From the softgoal example of user-friendliness, a common solution offered is that of graphical user interface (GUI). *Claim softgoals* argue the rationale and explain the context for a softgoal or interdependency link. Continuing with the small example of user-friendliness, the 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 are BREAK, HURT, NUKN, EQL, UKN, PUKN, HELP, and MAKE. As their names suggest, BREAK creates a sure negative effect, HURT imposes a partial negative effect, and NUKN may cause a negative effect. EQL mirrors the same effect, and UKN produces an unknown or neutral effect. PUKN many cause a positive effect, HELP triggers a partially positive effect, and MAKE ensures a totally positive effect. 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 exist for the goal to exist. OR defines the relationship that at least one of the refined subgoals must exist for the goal to exist.

2.3 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 determined according to the mappings of Table 1. In short, 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 combined effect from the AND, and OR decompositions is computed by equation 1. Intuitively, the AND function echoes the aphorism “a chain is as strong as its weakest link”, and

the OR function shows that its effect is as strong as its strongest link. For instance, all of the sources must be satisficed to produce a satisficed destination within the AND composition whereas only one satisficed source would yield a satisficed destination within the OR composition. Finally, all of the attributing effects are united to an ultimate effect according to equation 2. Intuitively, equation 2 is an extension of the aphorism underlined by the AND composition in so much that the overall united effect is "as strong as its weakest link."

		interdependency link						
	BREAK	HURT	NUKN	EQL	UKN	PUKN	HELP	MAKE
source label S	D	U+	U-, D	S	U	U+, S	U+	S
D	U+	U-	U+	D	U	U-	U-	U-
C	?	?	?	C	U	?	?	?
U	U	U	U	U	U	U	U	U

S = satisficed; D = denied; C = conflict; U = unknown + = positive, - = negative, ? = unknown

Table 1: The individual effect of source label upon destination given link label 'S'

$$AND(G0,\{G1,G2,\ldots,Gn\}: label(G0) = \min(labl(Gi))$$
$$OR(G0,\{G1,G2,\ldots,Gn\}: label(G0) = \max(label(Gi))$$
$$where S \geq \{U,C\} \geq D \qquad [1]$$

$$Unite: \min_{i \in I}(l)$$
$$where \{S,D\} \geq U \geq C \qquad [2]$$

2.4 Notation

The iconic notation used to denote the various constructs of the NFR framework are shown in Table 2. <explain in words each notation>

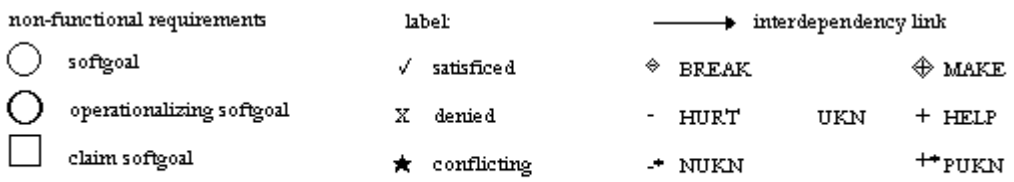

Table 2: NFR Framework notation

3.0 NFR-Assistant CASE Tool

The NFR-Assistant is presented as a Computer Aided Software Engineering (CASE) tool that incorporates the semantics and syntax of the NFR framework into a practical application to aid software developers in analyzing non-functional requirements. The iterative process of cataloging and analyzing customer requirements has the tendency to be time-consuming, and the complex network of non-functional requirements necessitates painstaking attention. The NFR-Assistant CASE tool facilitates this process with its pliable graphical representations, so as to demand minimal adjustments. Moreover, it operates the error-prone task of incrementally tracing the effects of the non-functional requirements and still endows the developer with full control over the decision-making process by providing a semi-automated, interactive evaluation procedure. Hence, the NFR-Assistant CASE tool strives to alleviate the repercussions within the dynamic world of software engineering.

3.1 Development

Since the NFR-Assistant CASE tool facilitates the decision-making process in any software development, a reflective case study was conducted for the NFR-Assistant CASE tool itself. To derive the non-functional requirements for the NFR-Assistant, consider the following typical usage scenario.

> *"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 directs constraints and resources. This collaborative process may iterate in any order or occur simultaneously, and all information is analyzed and recorded via the NFR-Assistant CASE tool."*

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 reference the CASE tool in a joint effort. In order to produce this collaborative groupware, a distributed network must be established. Leveraging tried 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 and methodology, extendibility and evolvability are paramount to the life of the NFR-Assistant. Thus, the set of non-functional requirements considered for the NFR Assistant Case tool consists of the following.

- Collaborativeness of groupware
  - Interactivity
  - Consistency of SIG
  - Distributedness of network
- Userfriendliness of HCI
- Extendibility
- Evolvability
- Reliability of communication
  - Integrity
  - Maintainability
  - Availability
    - Platform independence

Figure 1 extends these visually in addition to their correlation.

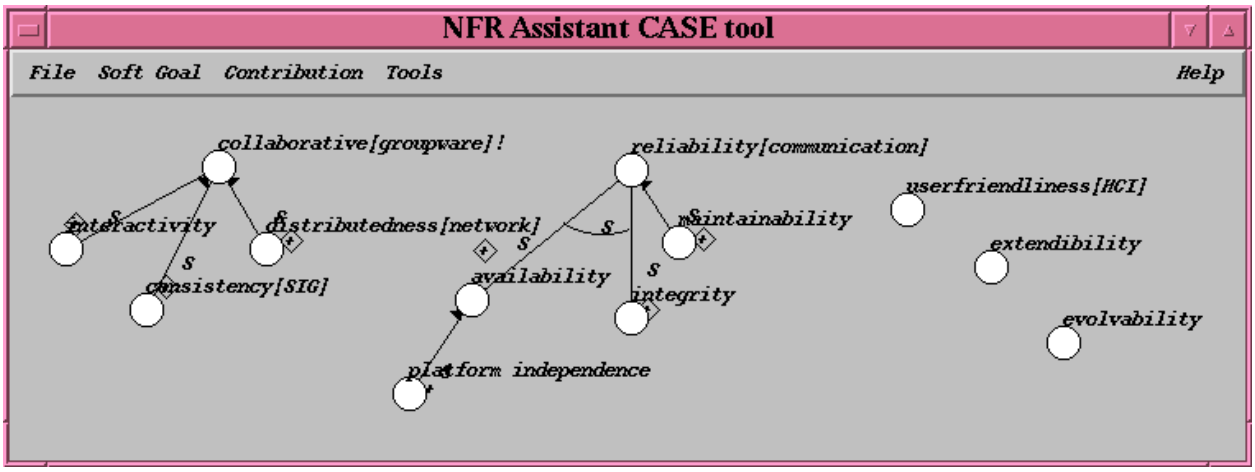

Figure 1: NFRs for the NFR-Assistant CASE tool

With these end goals in mind, the system architecture can then be decided upon rationally and systematically.

3.2 System Architecture

The fundamental concepts and control flow for the NFR-Assistant system architecture are 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.

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. Each software architecture scheme was analyzed within the context of the

NFR-Assistant CASE tool to expose its advantages and disadvantages. 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 2: Pipe & Filter architecture

The Pipe-and-Filter architecture (shown in Figure 2) provides a very simple, direct process. It is most appropriate in applications where completely independent tasks are to be performed in succession. However, it is not amendable 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 is probably not conducive to projections of distributedness.

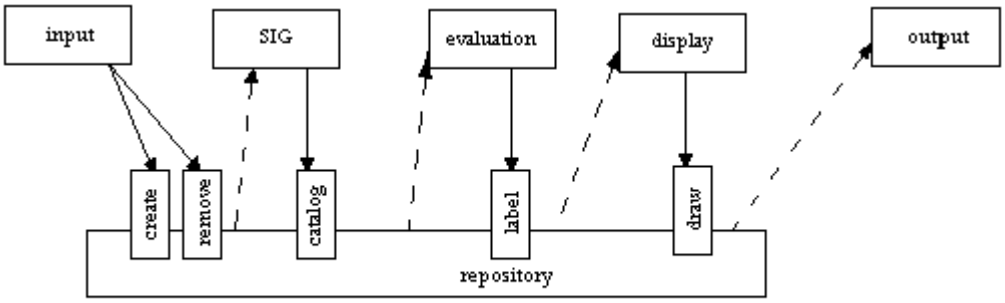

Figure 3: Implicit Invocation

The Implicit Invocation architectural style (shown in Figure 3) 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. Thus, 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. Its shared data structures within the *repository* module reduce the computational overhead of inter-process communication, and reduce the memory overhead. However, its consistency and/or interactivity may suffer when multiple users attempt to access the shared data as is anticipated by the usage scenario. Thus, 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 CASE tool.

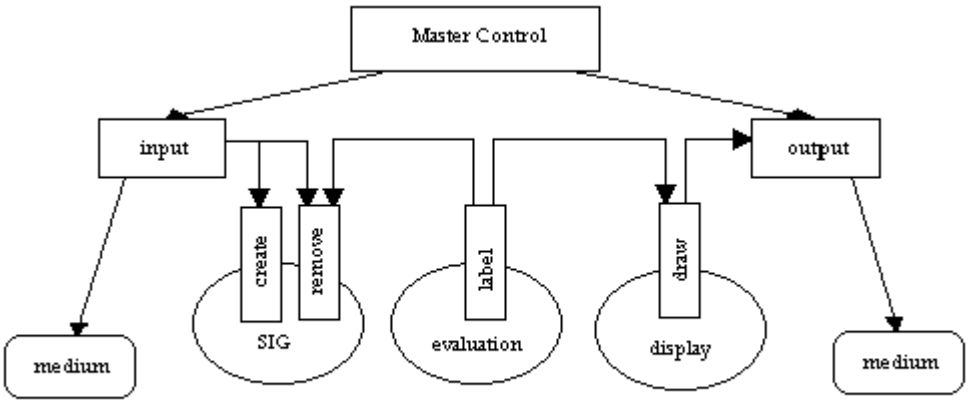

Figure 4: Abstract Data Type

The Abstract Data Type (ADT) architecture (shown in Figure 4) 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.

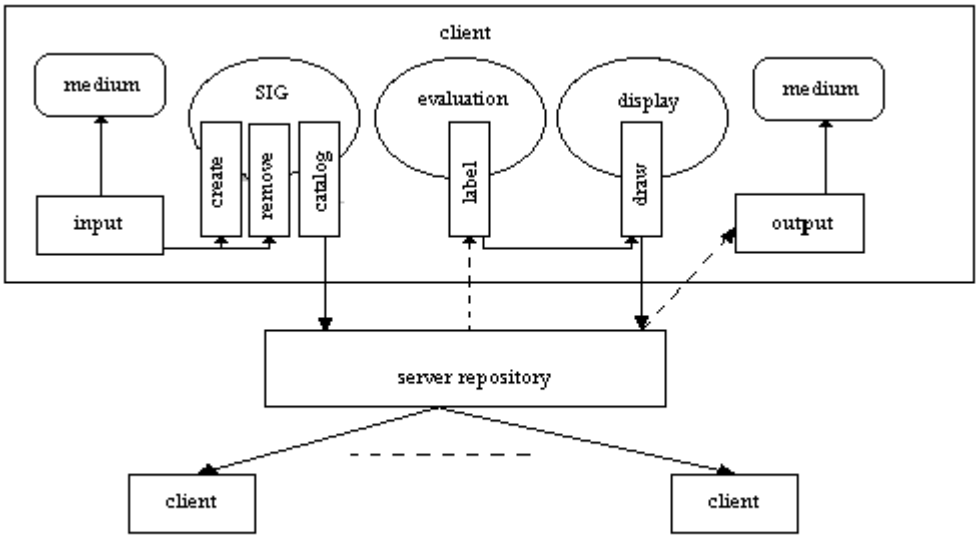

Figure 5: Web Client/Server ADT with Events

The *Web Client/Server Abstract Data Type with Events* architecture (as shown in Figure 5) 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. Throughout this decision process, the NFR-Assistant tool was implemented to verify our rationalization and to record the solution propositions.

Figure 6 shows the analysis results of the selected *Web Client/Server ADT with Events* architecture as well as a history of the discarded architecture alternatives. It is interesting to note that the softgoals *platform independence* and *user-friendliness[HCI]* remain unaffected by any of the proposed architectures.

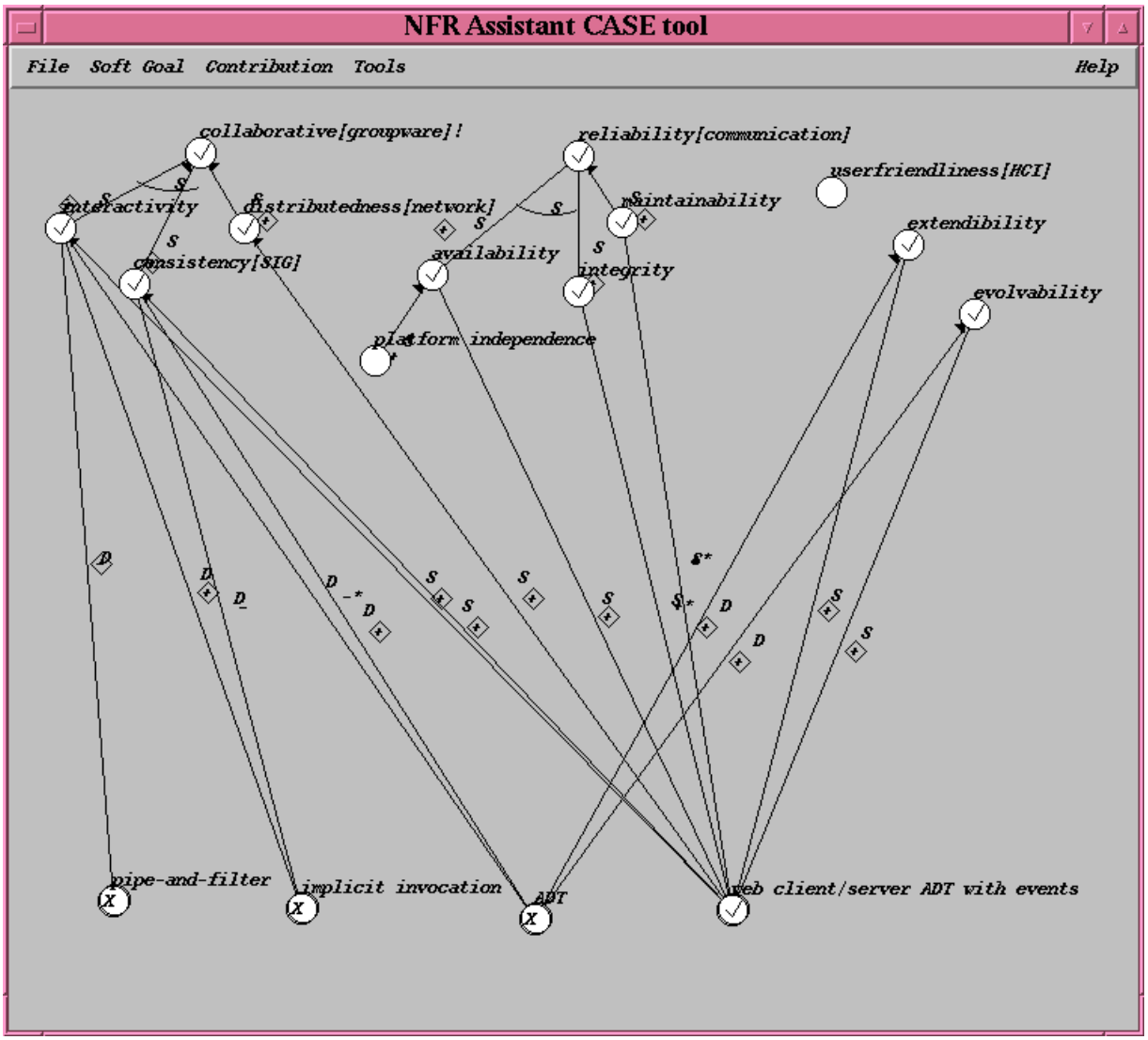

Figure 6: *SIG* of the NFR-Assistant CASE tool's system architecture decision process

3.3 Operationalizing Goal: Programming language

The current implementation of the NFR-Assistant is a Java applet [T98] that features a subset of the NFR framework. The programming language of Java was selected for the NFR-Assistant in a similar fashion to the decision of its architecture. The same set of softgoals was used to evaluate solution alternatives for languages such as Prolog, Visual Basic, and C++ with Tcl/Tk. Java's axiom of "write once, run everywhere" accomplishes the NFR-Assistant softgoal of

platform independence, and Java's savvy graphical user interface achieves the softgoal of *user-friendliness [HCI]*. Java has the advantage over the other languages in its availability via the world-wide-web and courtesy of its emerging predominance in the programming community. Moreover, Java serves as a good implementation of the *Web Client/Server ADT with Events* architecture paradigm. Java is defined upon an event-driven system, rooted in object-oriented programming, supported by the web, and may be configured within a client/server network.

4.0 Related Works

The services offered by the NFR-Assistant are similar in spirit to those of SYBIL [Lee92], which extends earlier work on decision support systems, in particular gIBIS [Conklin88]. 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 offers three types of softgoals to represent design constraints, techniques and rationale, which then facilitate the use of NFRs as criteria for selecting among design alternatives. The NFR-Assistant also offers generic methods and correlation rules for knowledge of design techniques and tradeoffs to be captured, encoded and reused. These methods and correlation rules, along with the labeling procedure, make it possible to automate parts of the design process.

The Java rendition of a subset of the NFR-Assistant 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 [JMSV92b]. 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 [MBJK91] to provide support for agent-oriented requirements analysis

[Y94].

The NFR-Assistant is also similar to recent work by Boehm and In [BI96] that explores a knowledge-based tool for identifying potential conflicts among NFRs early in the software / system life cycle. The work examines tradeoffs involved in software architecture and process strategies, in the context of the USC-CSE WinWin groupware support system for determining software and system requirements as negotiated win conditions. WinWin's emphasis in negotiation support is complementary to the NFR-Assistant's emphasis in the wide spectrum of informal, semi-formal and formal notation and methods.

The NFR-Assistant grows out of the earlier Mapping-Assistant of the DAIDA environment [JMSV92a] that provides support for all phases of information system engineering. The Mapping-Assistant [Chung92] 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 [Fickas87] [Fickas88] 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.

Quality Function Deployment (or *the House of Quality*) [HC88] in industrial engineering has been applied to Software Quality Assurance (e.g., [MS87]). 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 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. 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.

5.0 Conclusion

The main technical contribution is a Java rendition of a subset of the NFR-Assistant, 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.

Future work should 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 setting once a server is available. Instead of uploading independent copies of this Java applet, a connection could be established among users at various physical locations to allow collaboration on the NFR framework via the World Wide Web. 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.

Collection of feedback and preliminary analysis is being reviewed for the working prototype that is presently residing on the Internet. Since the NFR-Assistant CASE tool is a visualization mechanism to represent and manipulate non-functional requirements, it is governed by the same constraints 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 this nuance.

References

[BI96]
B. Boehm and H. In, "Identifying Quality-Requirements Conflicts", *IEEE Software*, March 1996.

[CNYM99]
L. Chung, B. A. Nixon, E. Yu and J. Mylopoulos, *Non-functional Requirements in Software ngineering*, Kluwer Publishing (to appear).

[CNYM99]
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.

[F85] S. F. Fickas, Automating the Transformational Development of Software. IEEE TSE, Vol. SE—11, No. 11, November 1985, pp. 1268—1277.

[HC88]
J. R. Hauser and D. Clausing, "The House of Quality,"*Harvard Business Review*, May—June 1988, pp. 63—73.

[JMSV92a]
M. Jarke, J. Mylopoulos, J. W. Schmidt, Y. Vassiliou, "DAIDA: An Environment for Evolving Information Systems," *ACM Trans. Information Systems*, vol. 10, no. 1, Jan. 1992, pp. 1—50.

[JMSV92b]
M. Jarke (Editor), *ConceptBase V3.1 User Manual*. Univ. of Passau, 1992.

[L91]
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.

[MS87]
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.

[MBJK91]
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.

[N97]
B. A. Nixon, *Dealing with Performance Requirements for Information Systems*. Ph.D. Thesis, Dept. of Computer Science, Univ. of Toronto, 1997.

[S81]
H. A. Simon, *The Sciences of the Artificial*, Second Edition. Cambridge, MA: The MIT Press, 1981.

[T98]
Q. Tran, "A CASE Tool for the Non-Functional Requirements Framework." M.S. Thesis, Dept. of Computer Science, Univ. of Texas Dallas, 1998.

[Y94]
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.