Chapter

Architectural Design to Meet Stakeholder Requirements

L. Chung', D. Gross" & E. Yu"
Computer Science Program, University of Texas, Dallas, USA' & Faculty of Information Studies, University of Toronto, Toronto, Ontario, Canada"

Key words: software architecture, rationale, stakeholders, organization modeling, requirements, quality attributes, architectural properties, non-functional requirements, process-oriented, softgoal, satisficing, design reasoning

Abstract: Architectural design occupies a pivotal position in software engineering. It is during architectural design that crucial requirements such as performance, reliability, costs, etc., must be addressed. Yet the task of achieving these properties remains a difficult one. Senior architects with many years of experience have to make difficult choices to meet competing requirements. This task is made even more difficult with the shift in software engineering paradigm from monolithic, stand-alone, built-from-scratch systems to componentized, evolvable, standards-based, and product line oriented systems. Many well-established design strategies need to be reconsidered as new requirements such as evolvability, reusability, time-to-market, etc., are becoming more important. These requirements do not come from a single source, but result from negotiations among many stakeholders. A systematic framework is needed to help architects achieve quality requirements during architectural design. This paper outlines an approach that formulates architectural properties such as modifiability and performance as “softgoals” which are incrementally refined. Tradeoffs are made as conflicts and synergies are discovered. Architectural decisions are traced to stakeholders and their dependency relationships. Knowledge-based tool support for the process would provide guidance during design as well as records of design rationales to facilitate understanding and change management.

1. INTRODUCTION

The importance of architectural design is now widely recognized in software engineering, as evidenced by the recent emergence of seminal reference texts e.g. (Shaw & Garlan, 1996; Bass, 1998) and several international workshop series and special sessions in major conferences. It is acknowledged, however, that many issues in software architecture are just beginning to be addressed. One key task that remains a difficult challenge for practitioners is how to proceed from requirements to architectural design.

This task has been made much more difficult as a result of today's changing software environment. Systems are no longer monolithic, built from scratch, or operate in isolation. Systems built in the old paradigm have contributed to the legacy system problem. Today's systems must be developed quickly, evolve smoothly, and interoperate with many other systems. Today's architects adopt strategies such as reusability, componentization, platform-based, standards-based, etc., to address new business level objectives such as rapid time-to-market, product line orientation, and customizability. Two important aspects may be noted in this shift in software engineering environment: (i) there have been significant shifts in architectural quality objectives; and (ii) architectural requirements are originating from a much more complex network of stakeholders.

System-wide software qualities have been recognized to be important since the early days of software engineering. For example, (Boehm, 1976) and (Bowen, 1985) classified a number of software attributes such as flexibility, integrity, performance, maintainability, etc. It is well known that these quality attributes (also referred to as non-functional requirements) are hard to deal with, because they are often ill defined and subjective. The recent flurry of activities on software architecture involving researchers and practitioners have refocused attention on these software qualities since it is realized that system-wide qualities are largely determined during the architectural design stage (Boehm, 1992; Perry, 1992; Kazman, 1994; Shaw & Garlan 1996; Bass, 1998). With the shift to the new, fast-cycled, component-oriented software environment, priorities among many quality objectives have changed, and new objectives such as reusability and standards compliance are becoming more prominent. While performance will continue to be important, it must now be traded off against many kinds of flexibility. As a result, many architectural solutions that were well accepted in the past need to be rethought to adapt to changes in architectural objectives.

When systems were stand-alone and had definite lifetimes, requirements could usually be traced to a small, well-defined set of stakeholders. In the new software environment, systems tend to be much more widely

interconnected, have a more varied range of potential customers and user groups (e.g., due to product line orientation), may fall under different organizational jurisdictions (at any one time, and also over time), and may evolve indefinitely over many incarnations. The development organization itself, including architects, designers, and managers, may undergo many changes in structure and personnel. Requirements need to be negotiated among stakeholders. In the case of architectural quality requirements, the negotiations may be especially challenging due to the vagueness and open-endedness of initial requirements. Understanding the network of relationships among stakeholders is therefore an important part of the challenge faced by the architect practitioner.

These trends suggest the need for frameworks, techniques, and tools that can support the systematic achievement of architectural quality objectives in the context of complex stakeholder relationships.

In this paper, we outline an approach which provides a goal-oriented process support framework, coupled with a model of stakeholder relationships. The paper includes simplified presentations of the NFR Framework (Chung, 1998) and the *i** framework (Yu, 1995). A web-based information system example, incorporating a KWIC component, is used to illustrate the proposed approach.

2. GOAL-ORIENTED PROCESS SUPPORT FOR ARCHITECTURAL DESIGN

Consider the design of a web-based information system. There would be a set of desired functionalities, such as for searching information, retrieving it, scanning it, downloading it, etc. There would also be a number of quality requirements such as fast response time, low storage, ease of use, rapid development cycle, adaptability to interoperate with other systems, modifiability to offer new services, etc. The functional side of the requirements are handled by many development methodologies, from structured analysis and design, to recent object-oriented methods. Almost all these methods, however, focus overwhelmingly, if not exclusively, on dealing with functional requirements and design. While there is almost universal agreement on the crucial importance of achieving the quality requirements, current practice is often ad hoc, relying on after-the-fact evaluation of quality attributes. Techniques for evaluating and assessing a completed architectural design (“product”) are certainly valuable. However, such techniques usually do not provide the needed step-by-step (“process”) guidance on how to seek out architectural solutions that balance the many competing requirements.

Complementary to the product-oriented approaches, the NFR Framework (Chung, 1993, 1998) takes a *process-oriented* approach to dealing with quality requirements. In the framework, quality requirements are treated as (potentially conflicting or synergistic) goals to be achieved, and used to guide and rationalize the various design decisions during the system/software development. Because quality requirements are often subjective by nature, they are often achieved not in an absolute sense, but to a sufficient or satisfactory extent (the notion of *satisficing*). Accordingly, the NFR Framework introduces the concept of *softgoals*, whose achievement is judged by the sufficiency of contributions from other (sub-) softgoals. Throughout the development process, consideration of design alternatives, analysis of design tradeoffs and rationalization of design decisions are all carried out in relation to the stated softgoals and their refinements. A *softgoal interdependency graph* is used to support the systematic, goal-oriented process of architectural design. It also serves to provide historical records for design replay, analysis, revisions, and change management.

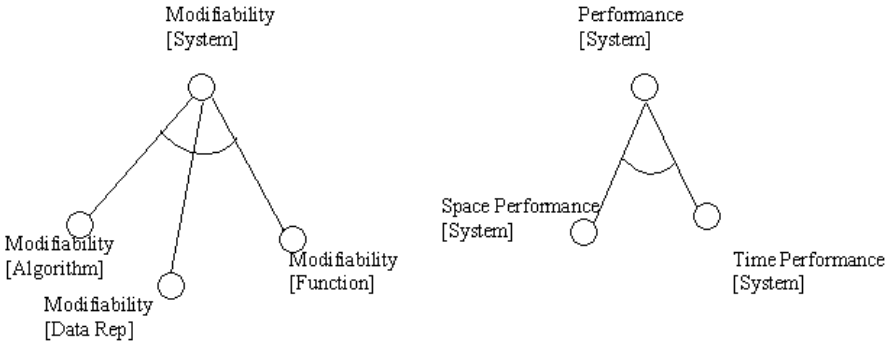

Figure 1. A softgoal interdependency graph showing refinements of quality requirements based on topic and type

For the purpose of illustration, let us consider a small part of the example in which a keyword in context (KWIC) system is needed. The KWIC system is part of a web information system, used to support an electronic-shopping catalog. Suppose the KWIC system architect is faced with an initial set of quality requirements: “the system should be modifiable” and “the system should have good performance”. In the aforementioned process-oriented approach, the architect explicitly represents each of these as a softgoal to be achieved during the architectural design process. Each softgoal (e.g., Modifiability [system]) is associated with a type (Modifiability) and a topic (system), along with other information such as importance, satisficing status and time of creation. Figure 1 shows the two softgoals as the top level nodes.

As these high level requirements may mean different things to different people, the architect needs to first clarify their meanings. This is done through an iterative process of softgoal refinement which may involve reviewing the literature and consulting with domain experts. After consultation, the architect may refine Modifiability [System] into three offspring softgoals: Modifiability [Algorithm], Modifiability [Data representation], and Modifiability [Function]. This refinement is based on topic, since it is the topic (System) that gets refined, while the softgoal type (Modifiability) is unchanged. This step may be justified by referring to the work by Garlan and Shaw (Garlan, 1993), who consider changes in processing algorithm and changes in data representation, and to Garlan, Kaiser, and Notkin (Garlan, 1992), who extend the consideration with enhancement to system function. Similarly, the architect refines Performance [System], this time based on its type, into Space Performance [System] and Time Performance [System], referring to work by Nixon (Nixon, 1993).

Figure 1 shows the two refinements. In the figure, a small "arc" denotes an "AND" *contribution*, meaning that in order to satisfice the parent softgoal, all of its offsprings need to be satisficed. As will be shown later, there are also other contribution types, including "OR" and partial positive (+) or negative (-) contributions. Contribution types are important for deciding the satisficing status of a softgoal based on contributions towards it.

In parallel to the refinement of quality requirements, the software architect will consider different ways of meeting the KWIC functional requirements in the context of the web information system. At various points during the design process, the architect will go through a number of interleaving activities of componentization, composition, choice of architectural style, etc. Each activity can involve consideration of alternatives, where NFRs can guide selection, hence narrowing down the set of architectural alternatives to be further considered.

For example, the architect can consider architectures with varying numbers of (main) components: i) Input, Circular Shift, Alphabetizer and Output; ii) Input, Line Storage, Circular Shift, Alphabetizer and Output and so forth. Each choice will make particular contributions to the NFRs. With either choice the architect can further consider alternatives about control, for example, one with a Master Control and one without. Yet another decision point might concern the way data is shared: sharing of data in the main memory, sharing of data in a database, sharing of data in a repository with an event manager and so forth. Figure 2 describe some of the above alternative architectures using "conventional" block diagrams. The diagrams were redrawn by one of the authors based on (Shaw & Garlan, 1996).

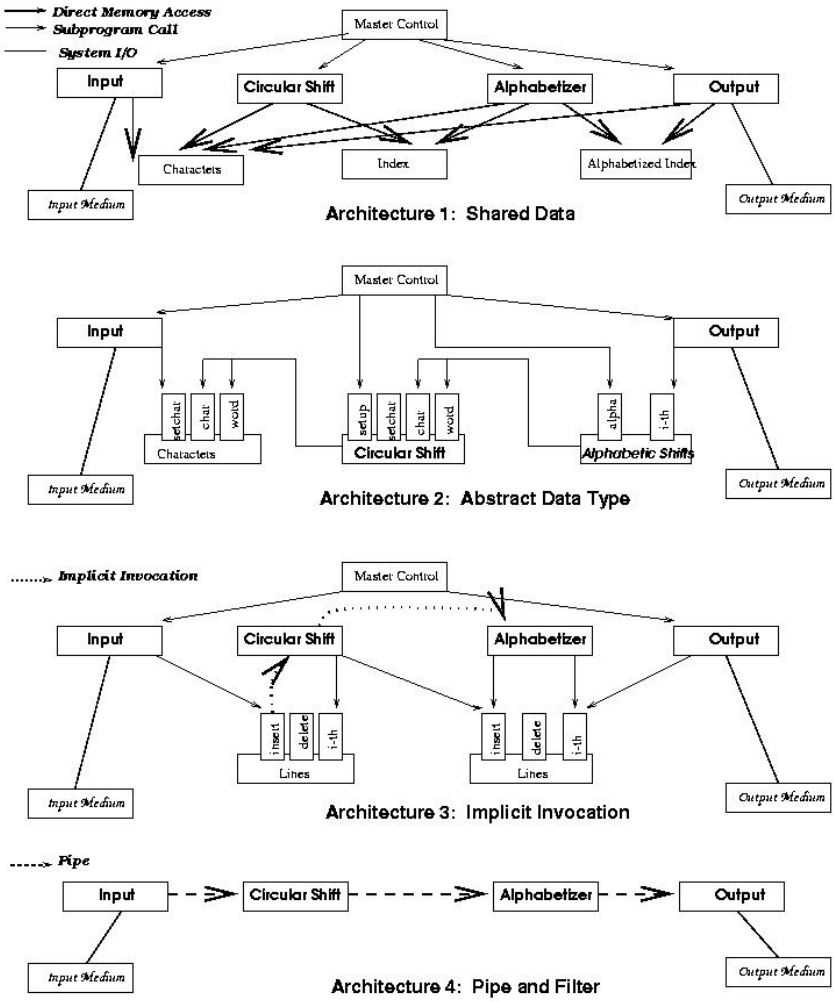

Figure 2. Architectural alternatives for a KWIC system

Let us assume that the architect is interested in an architecture which can contribute positively to the softgoal Modifiability [Data representation], and considers the use of an "Abstract Data Type" style of architecture, as discussed by Parnas (Parnas, 1972) , and Garlan and Shaw (Garlan, 1993): components communicate with each other by means of explicit invocation of procedures as defined by component interfaces.

As the architect would learn sooner or later, the positive contribution of the Abstract Data Type architecture towards modifiable data representation

is made at the expense of another softgoal, namely the time performance softgoal. Figure 3 shows the positive contribution made by the abstract data type solution by means of "+" and the negative contribution by "-" *contribution link.*

The architect would want to consider other architectural alternatives in order to better satisfice the stated softgoals. The architect may discover from the literature that a "Shared Data" architecture typically would not degrade system response time, at least when compared to the Abstract Data Type architecture, and more importantly perhaps it is quite favorable with respect to space requirements. This discovery draws on work by Parnas (Parnas, 1972), and by Garlan and Shaw (Garlan, 1993) who considered a Shared Data architecture in which the basic components (modules) communicate with each other by means of shared storage. Not unlike the Abstract Data Type architecture, however, the Shared Data architecture also has some negative influence on several other softgoals: a negative (-) impact on modifiability of the underlying algorithm (process) and a very negative (--) impact on modifiability of data representation.

Figure 3 shows both design steps along with the various contributions that each alternative makes towards the refined softgoals. Note that the diagram is build iteratively rather than in one step -- according to the architectural "discovery process" of the architect.

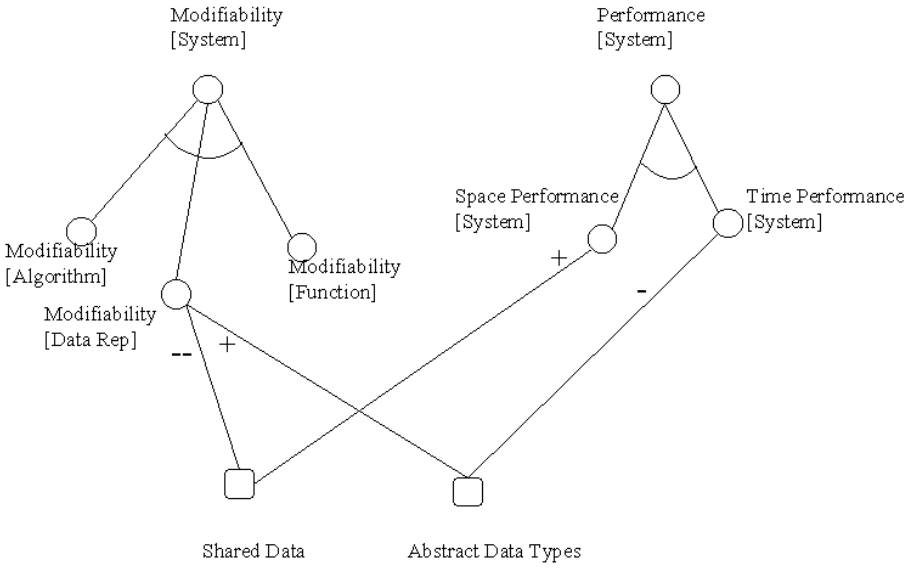

Figure 3. Contribution of the Shared Data and Abstract Data Type architectures

Interestingly, Figure 3 shows tradeoffs between the architectural alternatives that have been considered so far. The architect can continue to consider other architectural alternatives, including hybrid solutions, or decide which of the two better suits the needs of the stakeholders. How can the architect go about doing the latter, if that is what she so desires? One way to do the tradeoff analysis is by using the degree of *criticality* (or priority, or dominance, or importance) of the quality requirements. In the context of a particular web information system, for example, the stakeholders might indicate that performance is more critical than modifiability. In this case, then, the architect would choose Shared Data over Abstract Data Type, since Shared Data is more satisfactory with respect to both space and time performance, hence the overall performance requirements (recall the "AND refinement").

During the process of architecting, the architect needs to make many decisions, most likely in consultation with stakeholders. As the above discussion suggests, an interesting question is: "how can the architect evaluate the impact of the various decisions?" The NFR Framework provides an interactive evaluation procedure, which propagates labels associated with softgoals representing their satisficing status (such as *satisficed, denied, undetermined, and conflict*) across the softgoal interdependency graph. Labels are propagated along the direction of contribution, usually "upwards" from specific, refined goals towards high level initial goals.

Because of the subjective nature of quality requirements, the software architect will want to explain and justify her decisions throughout the softgoal refinement process. This can be done in the NFR Framework using "*claims*". Claims can be attached to contributions (links in the graph) and to softgoals (nodes). Claims can themselves be justified by further claims. These rationales are important for facilitating understanding and evolution. For example, Shared Data may by and large have advantage over Abstract Data Type with respect to space consumption. This general relationship, however, may need to be argued for (or against), in the context of the particular web information system. If, for example, the volume of the data to be maintained by the system is low, the relative advantage of Shared Data may not matter much. If this is indeed the case, the expected data volume can then be used as a claim against the relationship: "Shared Data makes a strong positive (++) contribution towards meeting space requirements". This might then lead the architect to choose Abstract Data Type as the ultimate architecture.

Figure 4 shows a softgoal interdependency graph for the KWIC system, taken from work by Chung, Nixon and Yu (Chung, 1995) which is based on (Garlan, 1993) and Garlan, Kaiser, and Notkin (Garlan, 1992).

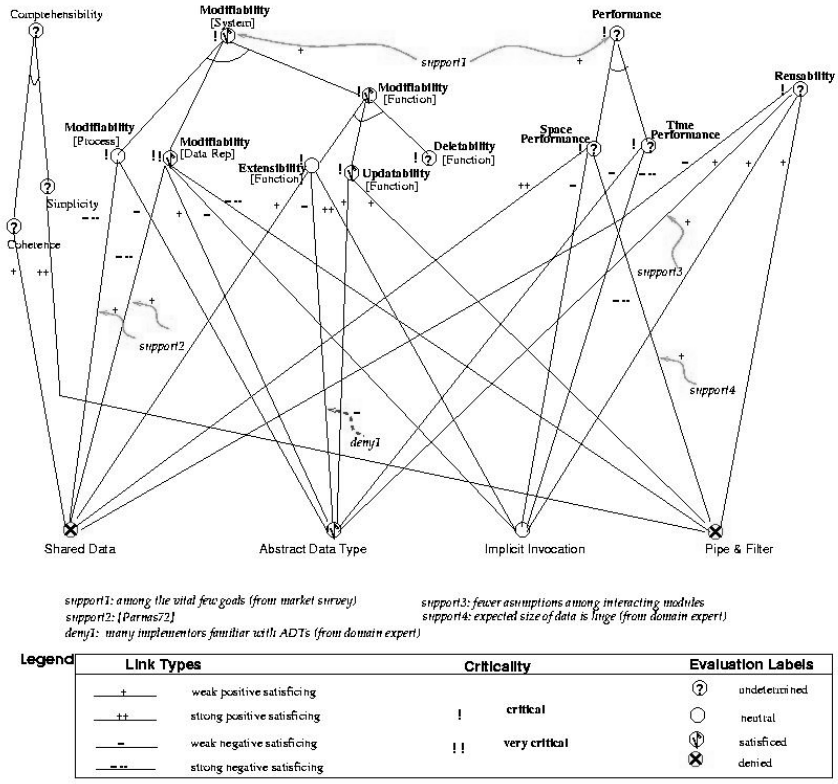

Figure 4. A softgoal interdependency graph for the KWIC system

3. MEETING DIFFERENT STAKEHOLDER REQUIREMENTS

We now illustrate the need to relate organizational context to the process, and consequently the outcomes, of architectural design. The illustration will be done through three scenarios, which will show that different sets of stakeholder concerns are transformed by the architectural design process into different architectural choices for information systems. More specifically, each different set of stakeholders and their concerns leads the architects to reason about different quality concerns, make and evaluate different design decisions, and finally leads, in our case, to the most appropriate architectural designs to be used in a particular web-based information system context.

3.1 Scenario 1

An *e-shopping software vendor* specializes in offering software products which can be used in advertising, selling, and shipping goods and services in an Internet-based virtual market. The products should generate, among other things, e-catalogs so that any internet user can search for goods using a web-browser. The *e-catalog architect* realizes that she needs a software system which can generate an index, here an alphabetized list of the words in the descriptive text of each catalog item such that each word in the list is associated with a list of all catalog items pertaining to that word. Such a list, however, is just what a KWIC system generates. Hence, the *e-catalog architect* asks a *KWIC component architect* to built an indexing system. This is a brief description of the essential functional aspect of the scenario. We will shortly describe the quality aspect of the scenario, along with more details of the functional aspect.

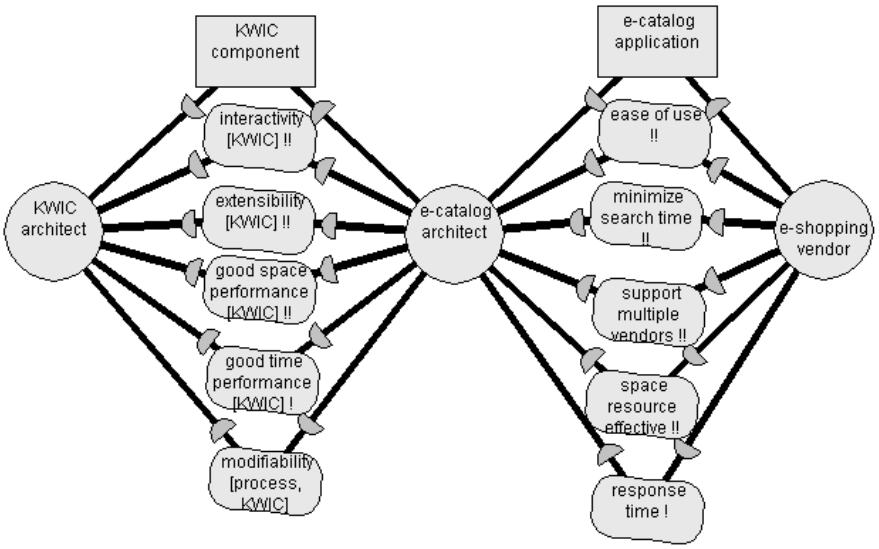

Figure 5. Organizational context for the e-catalog application

Figure 5 depicts the relationships among the three types of stakeholders, using the *i** framework proposed by Yu (Yu, 1994). The *i** framework allows for the description of *actors* and their *dependencies* in organizational settings. A circle represents an actor (e.g., *e-shopping vendor*) who may be dependent on some other actor (e.g., *e-catalog architect*) to achieve some of its goals (e.g., developing an e-catalog application). Not unlike the NFR Framework, the *i** framework also distinguishes a quality requirement,

denoted by a cloud like shape (to suggest softness), from a functional one, denoted by a rectangle. In the *i** framework , a dependency is described by a directed link between two actors. This type of graph is called a Strategic Dependency model in the *i** framework (the other type of graph in *i** -- the Strategic Rationale model will not be discussed in this paper).

In the current scenario, the *e-shopping vendor* depends on the *e-catalog architect* to deliver an e-catalog application, who in turn depends on the *KWIC component architect* to deliver an indexing system.

This kind of diagram shows where requirements originate. It also serves as a basis for determining what kind of negotiated delegations should take place, how different architectural decisions affect the various stakeholders, and possibly what kind of requirements to allocate to, and how to partition the system into, sub-systems and components. Just like a softgoal interdependency graph, it becomes a basis for justification and system/software architectural evolution.

Now we describe the quality concerns of the stakeholders. To start with, the e-shopping vendor expects the application software system to be easy to use. The vendor also has other concerns. As the catalog items is expected to grow quite rapidly, storage space resource is a very important concern, as is fast response time. Also shown in Figure 5 is a multiple-vendor support, namely, allowing for the integration of catalogs that reside on various server machines in physically remote vendor organizations. The exclamation marks denote the criticality of a quality. The highest priority is assigned to two exclamation marks, medium priority to one, and low priority none.

As a matter of fact, the list of quality requirements and their criticalities is determined through cooperation between the e-catalog architect and the e-shopping vendor who go through a process of recursive refinements, in the manner of the previous section, which may also require the KWIC component architect's involvement at least occasionally. The list then becomes what is commonly known as the user requirements.

When the user requirements are more or less satisfactory, the e-catalog architect directs her attention more towards defining the system requirements, whose clarification may need more of the KWIC component architect's involvement than before. The system requirements may inherit some of the user requirements more or less directly, such as good space and response time requirements. The system requirements will also come from the system's perspective. For example, the "ease of use" requirement now may be translated more specifically into interactivity (such as configuring indexing options dynamically) and extensibility (such as allowing for the use of international language character sets, categorical search and phonetic search). Another system requirements that might be considered is the modifiability requirement, here for changing the overall algorithm which

builds those indices transparently in a distributed setting. The criticalities may also change, due to the new requirements and the derived requirements. For example, in the presence of the extensibility requirement, which is new, the criticality of the good time performance requirement is lowered from critical to medium.

With the organizational context in place, the KWIC component architect uses the process-oriented NFR Framework to refine the quality softgoals, consider architectural design alternatives, carry out tradeoff analysis and evaluate the degree to which softgoals are satisficed, all in consideration of the context. The top portion of Figure 6 represents those softgoals that originated from the e-shopping vendor, and are negotiated and delegated through the e-catalog architect to the KWIC architect. The relative criticality values are preserved in the softgoal interdependency graph. Figure 6 shows the result of the process whereby the architect has arrived at four architectural alternatives in an attempt to satisfice the stated softgoals.

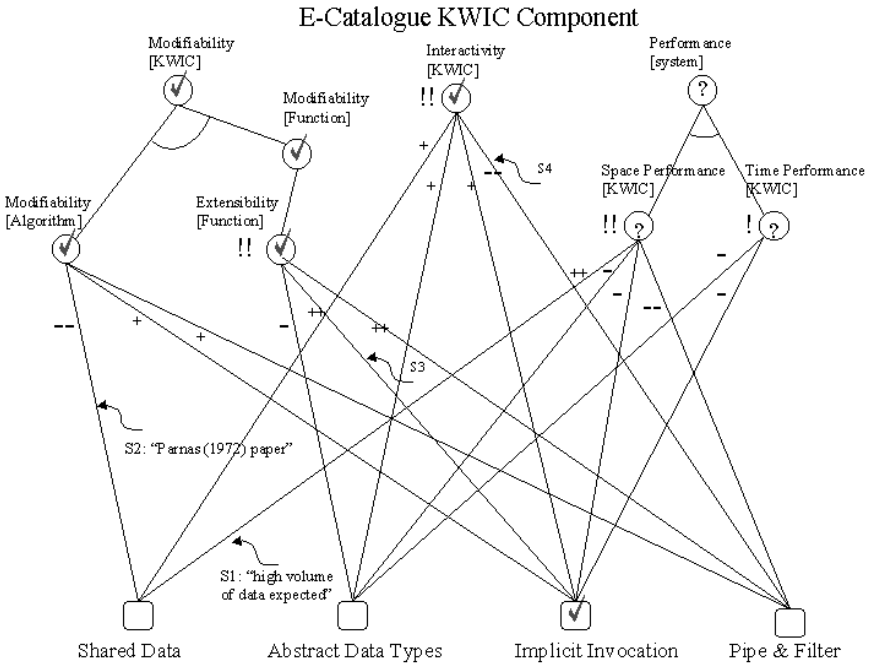

Figure 6. A softgoal interdependency graph for the e-catalog KWIC Component

Importantly, the diagram in figure 6 shows a number of claims, which derive from the knowledge of the organizational context, and which are used to argue for, or against, the types of softgoal criticalities and interdepencies, and consequently in softgoal evaluation and selection among architectural

alternatives. For example, using the Shared Data architectural style is expected to have a very good contribution towards space performance. The architect uses the organizational context diagram (figure 5) to find some argument in support (or denial) of that particular contribution. In the current scenario, for example, the architect argues for the validity of the contribution by pointing to the e-shopping vendor who wants the system to have the ability to handle a rapidly growing number of catalog items. This claim is denoted by the "S1" arrow in figure 6.

Despite the significant savings by the Shared Data architecture in data storage, however, the Implicit Invocation architecture seems to be the most promising for achieving extensibility of function, which is as critical as space performance. Furthermore, Implicit Invocation helps modifiability of processes, in contrast to Shared Data, although there is a tie between the two concerning interactivity. Although not well met by Implicit Invocation, time performance is of low criticality. Taking all these into account, the KWIC architect chooses the Implicit Invocation as the target architectural design.

3.2 Scenario 2

A system administrator wants to offer the user a help facility which can retrieve all the documents that have some keyword in their description, as indicated by the user. The administrator, thus, asks a system architect to build such a help facility. The system architect, in turn, asks a KWIC component architect for an indexing software system, after realizing that the facility is essentially a KWIC system such as used the Unix "man –k" command.

Similar to figure 5 for scenario 1, we may now describe the three types of stakeholders using the *i** framework, together with the functional and quality requirements that the stakeholders delegate among themselves, together the various criticalities of each of the requirements. And analogous, to figure 6 for scenario 1, the architect iteratively builds an NFR softgoal interdependency graph in which she further refines the various quality requirements and argues for or against certain claims. These analogous figures for scenario 2 are not shown for lack of space, but some fragments of the functional and quality requirements as well as the (soft) goal interdependencies related to this scenario appear in figure 7 and 8.

Taking all contributions of each architectural style into account, together with the various criticalities of the softgoals to be achieved, the architect might want to choose the Pipe and Filter architectural style as the most promising one.

3.3 Scenario 3

A reuse manager is appointed by product line management to oversee the development of various systems in the organization. As it happens, the KWIC architect, the e-catalog architect and the help file system architect all work in the same organization. The reuse manager asks the KWIC architects to consider reuse as a critical priority and to maximize reuse of all components developed in that organization.

This scenario is especially interesting as it introduces a stakeholder (the reuse manager) whose global quality concern of having reusable components prompts the KWIC architect to find a solution that represents the union of quality concerns of all other architects, as well as taking into account each of their intended customer (the e-shopping vendor and the man administrator).

Essentially, figure 7 shows a merge of all stakeholders' quality softgoals discussed in the previous scenarios. In addition it show that reuse manager depends on the KWIC system architect to build a system that delivers and maximizes the use of reusable components for all development activities in that organization. Not shown are product line management stakeholders, who depend on the reuse manager for reduced development costs.

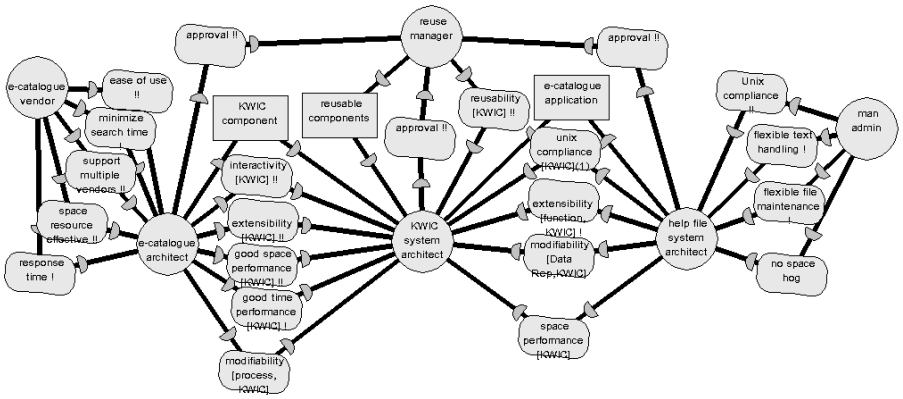

Figure 7. Organizational context for the Reuse requirement

For each of the two previous scenarios a different architectural solution style was chosen according to the specific kind of organizational context and its derived set of requirements. To find a reusable component solution the KWIC component architect will need to re-negotiate the delegated requirements with each of the involved stakeholders to overcome the stakeholders conflicting requirements. Perhaps the KWIC architect will also need to renegotiate the degree of reusability with the reuse manager.

Now that the architect has an organizational understanding (of which quality requirements and criticalities originated from which stakeholders,

and what network of relationships exists among the stakeholders), the architect now proceeds to use the NFR framework to evaluate, and further argue for or against the various architectural styles. During the evaluation, the architect renegotiates conflicting quality requirements and criticalities with the affected stakeholders and finds an architectural solution that makes acceptable trade-offs. Figure 8 shows the result of the architectural design process. (The "broken" lines are not part of the NFR Framework graphical notation, but are used in this paper to avoid cluttering the diagram with links not directly related to the architectural styles shown to be evaluated. The "e" subscript stands for the e-catalog architects point of view, while the "h" subscript stands for help file system architect's point of view).

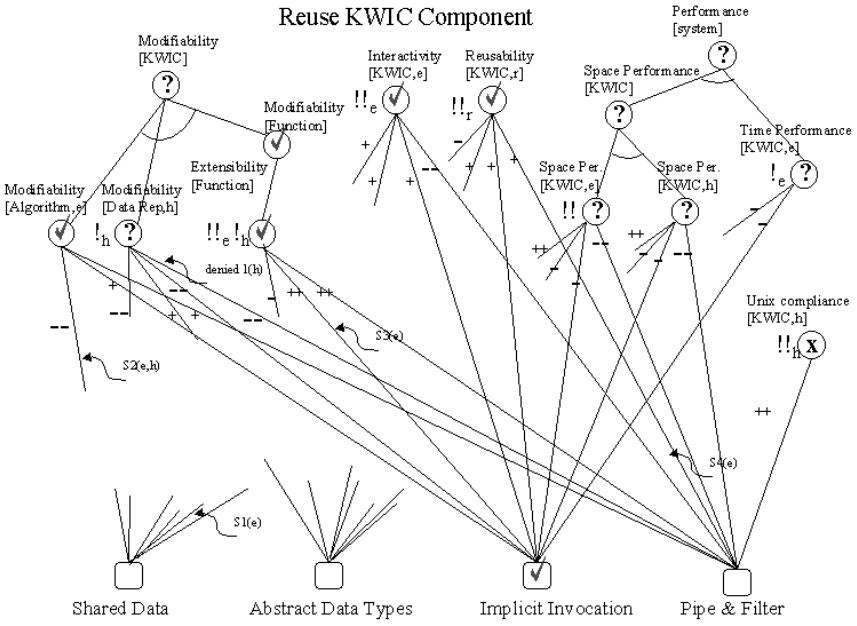

Figure 8. A softgoal interdependency graph for the Reuseable KWIC Component

The figure shows the architect evaluating the Implicit Invocation style for meeting the quality requirements originating from the e-catalog architect, the help file system architect and the reuse managers. While evaluating the Implicit Invocation style the architect may renegotiate with the help file system architect her demand for "Unix compliance" which, for her, would be better dealt with when using the Pipe & Filter style. The organizational context (such as the “approval” dependency that the architects have on the

reuse manager), will make the negotiating parties more forthcoming when concessions to their requirements and/or criticalities are needed.

4. DISCUSSION AND RELATED WORK

As pointed out by Garlan and Perry (Garlan, 1994), architectural design has traditionally been largely informal and ad hoc. Our proposal is aimed at rectifying some of the manifested symptoms by taking a more disciplined approach to architectural design. In particular, our proposal is aimed at improving our ability to understand the rationales behind architectural choices, hence making the system more easily traceable and evolvable. We have illustrated how to carry out a finer-grained analysis, and the comparison of architectural designs by considering quality-related concerns of multiple stakeholders and their interdependencies.

Our proposal draws on concepts that have been identified as essential to portray architectural infrastructure, such as elements, components, and connectors as suggested by Perry and Wolf (Perry, 1992), Garlan and Shaw (Garlan, 1993), Abowd, Allen, and Garlan (Abowd, 1993), and Robbins, Medvidovic, Redmiles and Rosenblum (Robbins, 1998). In our view, our emphasis on quality concerns and stakeholder interdependencies are complementary to efforts directed towards identification and formalization of concepts for functional architectural design.

Concerning the role of quality requirements, design rationale, and assessment of alternatives, the proposal by Perry and Wolf (Perry, 1992) is of close relevance to our work. Perry and Wolf propose to use architectural style for constraining the architecture and coordinating cooperating software architects. They also propose that rationale, together with elements and form, constitute the model of software architecture. In our approach, weighted properties of the architectural form are justified with respect to their positive and negative contributions to the stated NFRs, and weighted relationships of the architectural form are abstracted into contribution types and labels, which can be interactively and semi-automatically determined.

Boehm (Boehm, 1992), and Kazman, Bass, Abowd, and Webb (Kazman, 1994) have argued convincingly for the importance of addressing quality concerns in software architectures. Kazman, Bass, Abowd, and Webb (Kazman, 1994) propose a basis (called SAAM) for understanding and evaluating software architectures, and gives an illustration using modifiability. This proposal is similar to ours, in spirit, as both take a qualitative approach, instead of a metrics approach, but differs from ours

since SAAM is product-oriented, i.e., they use quality requirements to understand and/or evaluate architectural products.

In comparing architectural alternatives, it is intuitively appealing to use a tabular format. For example, in (Garlan & Shaw, 1993), a table is used to present the quality evaluations of four architectural alternatives. Such a table can be interpreted as depicting contributions from the architectural alternatives to the quality attributes treated as goals. In our study, we illustrated the importance of context and the need to trace design decisions to stakeholder requirements. Our approach suggests that the tabular representation of design alternatives and quality attributes is not sufficiently expressive.

We might consider extending the tabular representation by distinguishing quality requirements that come from different stakeholders, and by adding more explanatory notes such as the claims in the softgoal interdependency graphs.

Our approach emphasizes explicitly representing and using the quality concerns of multiple interacting stakeholders during the design of software architectures. Our approach is thus similar to the on-going work by Boehm and In (Boehm, 1996), who explore a knowledge-based tool for identifying potential conflicts among quality concerns early in the software/system life cycle, and using quality requirements in examining tradeoffs involved in software architectural design. Stakeholders such as user, maintainer, developer, customer, etc., are mapped to quality attributes in a graph. Our approach goes further by indicating that stakeholder requirements can be traced through a network of dependency relationships in an organizational model.

5. CONCLUSIONS AND FUTURE WORK

Achieving architectural quality requirements is a key objective in architecture-based approaches to software engineering. Quality requirements vary according to context and need to be negotiated among stakeholders. We have outlined a systematic approach for representing and addressing quality requirements during architectural design. The design reasoning is related to context through an organization model of stakeholder dependencies.

Using an extended version of the familiar KWIC example, we have illustrated how architectural decisions might vary depending on context, and how the design process can be guided and assisted using appropriate notational and reasoning support. The historical records of design decisions and rationales will facilitate understanding and evolution.

We have been working on tools to support the approach. These include facilities for generating and maintaining the graphs, for propagating labels, and for design revision. Knowledge for addressing specific quality requirements are codified in knowledge bases to assist in the refinement of goals. Known interactions among quality requirements are codified as correlation rules for detecting conflicts and synergies.

This paper represents a first step in an attempt to provide a systematic architectural design support framework that takes organizational and stakeholder relationships into account. We have drawn on the NFR framework for dealing with software quality requirements, and the *i** framework for modelling and reasoning about strategic actor relationships. In future work, we intend to further elaborate on issues specific to architectural design, and to better integrate architectural design reasoning and organizational relationships reasoning.

REFERENCES

Abowd, G., Allen R. and Garlan, D.(1993) "Using Style to Understand Descriptions of Software Architectures", *Software Engineering Notes*, 18(5): 9--20, *Proc. of SIGSOFT `93: Symposium on the Foundations of Software Engineering.*

Boehm, B. W. (1976) "Software Engineering", *IEEE Transactions on Computers*, 25(12), pp. 1226-1241

Bass, L., Clements P. and Kazman, R. (1998) Software Architecture in Practice, *SEI Series in Software Engineering*, Addison-Wesley.

Boehm, B. and Scherlis, B(1992) "Megaprogramming", *Proc. the DARPA Software Technology Conference*.

Boehm, B. and In, H.(1996) "Aids for Identifying Conflicts Among Quality Requirements", *Proc. International Conference on Requirements Engineering*, (ICRE96), Colorado, April 1996, and *IEEE Software*, March 1996.

Bowen, T. P. , Wigle, G. B. and Tsai, J. T. (1985) "Specification of Software Quality Attributes", Report RADC-TR-85-37, vol. I (Introduction), vol. II (Software Quality Specification Guidebook), vol III (Software Quality Evaluation Guidebook), Rome Air Development Center, Griffiss Air Force Base, NY, Feb. 1985.

Chung, L.K.(1993) "Representing and Using Non-Functional Requirements: A Process-Oriented Approach". *Ph.D. Thesis, Dept. of Computer Science, Univ. of Toronto*, June 1993. Also Technical Report DKBS--TR--93--1.

Chung, L.K. Nixon, B. and Yu, E.(1995) "Using Non-Functional Requirements to Systematically Select Among Alternatives in Architectural Design", *Proc., 1st Int. Workshop on Architectures for Software Systems*, Seattle, April 24-28, 1995., pp. 31-43.

Chung, L.K. Nixon, B. A., Yu, E and J. Mylopoulos(1998), Non-Functional Requirements in Software Engineering, Kluwer Publishing (to appear).

Garlan D. and Shaw, M.(1993) "An Introduction to Software Architecture *Advances in Software Engineering and Knowledge Engineering: Vol. I*, World Scientific Publishing Co.

Garlan, D., Kaiser, G. E. and Notkin, D. (1992) "Using Tool Abstraction to Compose Systems", *IEEE Computer*, Vol. 25, June 1992. pp. 30--38.

Garlan, D. and Shaw, M. (1993) “An Introduction to Software Architecture", *in Advances in Software Engineering and Knowledge Engineering: Vol. I,* World Scientific Publishing Co.

Garlan, D. and Perry, D.(1994) “Software Architecture: Practice, Potential, and Pitfalls", *Proc. 16th Int. Conf. on Software Engineering*, pp. 363--364.

Kazman, R, Bass, L., Abowd, G. and Webb, M. (1994) “SAAM: A Method for Analyzing the Properties of Software Architectures", *Proc. Int. Conf. on Software Engineering*, May 1994, pp. 81--90.

Nixon, B. A.(1993) “Dealing with Performance Requirements During the Development of Information Systems.", *Proc. IEEE Int. Symp. on Requirements Engineering*, San Diego, CA, January 4--6, Los Alamitos, CA: IEEE Computer Society Press, pp. 42--49.

Parnas, D. L. (1972) “On the Criteria to be Used in Decomposing Systems into Modules", *Communications of the ACM*, Vol. 15, Dec. 1972, pp. 1053--1058.

Perry, D. E. and Wolf, A. L. (1992) “Foundations for the Study of Software Architecture", *ACM SIGSOFT Software Engineering Notes*, 17(4), pp. 40--52.

Robbins, J. E. , Medvidovic, N., Redmiles, D. F. and Rosenblum, D. S. (1998) “Integrating Architecture Description Languages with a Standard Design Method", *Proc. 20th Int. Conf. on Software Engineering*, pp. 209--218.

Shaw, M. and Garlan, D. (1996) “Software Architecture: Perspectives on an Emerging Discipline”, Prentice Hall.

Yu, E. S. K. and Mylopoulos, J. (1994) “Understanding `”Why'” in Software Process Modelling, Analysis, and Design.”, *Proc., 16th Int. Conf. on Software Engineering*, Sorrento, Italy, May 1994, pp. 159--168.

Yu, E.(1995) “Modelling Strategic Relationships for Process Reengineering”, *Ph.D. Thesis, Dept. of Computer Science, Univ. of Toronto*.

APPENDIX

The KWIC problem statement (Parnas, 1972): “The KWIC [Key Word in Context] index system accepts an ordered set of lines; each line is an ordered set of words, and each word is an ordered set of characters. Any line may be “circularly shifted” by repeatedly removing the first word and appending it at the end of the line. The KWIC index system outputs a list of all circular shifts of all lines in alphabetical order.”