A Quality-Driven Systematic Approach for Architecting Distributed Software Applications

Tariq Al-Naeem[1], Ian Gorton[2], Muhammed Ali Babar[12], Fethi Rabhi[3] and Boualem Benatallah[1]
[1] School of Computer Science & Engineering, University of New South Wales, Australia; {tariqn,malibaba,boualem@cse.unsw.edu.au}
[2] National ICT Australia Ltd; {ian.gorton@nicta.com.au}
[3] School of Information Systems, Technology and Management, University of New South Wales, Australia; {f.rabhi@unsw.edu.au}

UNSW-CSE-TR-0433

September 2004

Abstract

Architecting distributed software applications is a complex design activity. It involves making decisions about a number of inter-dependent design choices that relate to a range of design concerns. Each decision requires selecting among a number of alternatives; each of which impacts differently on various quality attributes. Additionally, there are usually a number of stakeholders participating in the decision-making process with different, often conflicting, quality goals, and project constraints, such as cost and schedule. To facilitate the architectural design process, we propose a quantitative quality-driven approach that attempts to find the best possible fit between conflicting stakeholders' quality goals, competing architectural concerns, and project constraints. The approach uses optimization techniques to recommend the optimal candidate architecture. Applicability of the proposed approach is assessed using a real system.

1. Introduction

Designing the software architecture (SA) for a distributed application is widely an important and complex activity. Its importance lies in the fact that it embodies several design decisions that will be difficult and costly to change downstream if they are subsequently discovered to be flawed. It is complex because the architect must make complex design trade-offs to meet competing architectural requirements [1].

Further difficulties arise due to the early life-cycle nature of architecture design. As few or no concrete artifacts typically exist at this stage, it is hard, often impossible, to thoroughly reason about the consequences of many design decisions. This is especially true of design decisions that are embodied in off-the-shelf distributed component infrastructures (e.g. J2EE, .NET, CORBA) that are often utilized in the architecture. For these reasons, architects use prototypes and previous experience to justify their designs.

Additionally, there are often a number of stakeholders involved in the design process with each having conflicting quality goals. Furthermore, software design is often constrained by project cost and schedule, which always need to be satisfied.

In an attempt to help alleviate architectural design complexity, this research aims at providing techniques and tools to support disciplined reasoning during the SA design process. This work is motivated by the need to:

- Help various stakeholders of a system express the desired quality goals in a measurable form, and formalize the process of prioritizing those goals.
- Help architects determine the candidate architecture that best satisfy stakeholders' quality goals and meet stated project constraints.

This paper describes a quality driven design approach, *ArchDesigner*, that promotes a disciplined engineering and reasoning framework during SA design. The novelty of our approach lies in the use of optimization techniques, particularly Integer Programming [2], for optimizing the SA design comprised of multiple inter-dependent design decisions.

ArchDesigner improves upon previous approaches, which evaluate and select among given coarse-grained SAs [3, 4] without giving guidance on how to arrive at these architectures. We argue that the evaluation of all candidate SAs is a difficult, often impossible task, since the number of candidate SAs can be very large. Our approach therefore evaluates and selects among candidate SAs in a fine-grained fashion, thereby helping stakeholders arriving at a suitable SA solution.

We assess the applicability of ArchDesigner using a case study of a deployed system that had several stakeholders with different quality goals, multiple inter-dependent design decisions, and project constraints.

2. Background

Software quality is the degree to which an application possesses the desired combination of quality attributes [5]. SA plays a central role in achieving system wide quality attributes. The design of distributed application architectures is, however, inherently more complex than standalone systems. Distributed applications must deal with, for example, additional failure modes, non-determinism, deployment configuration and management, machine and network performance and scalability. It is consequently necessary to consider quality requirements early in the SA design stage.

2.1. Supporting quality considerations during SA design

The software engineering community has developed different methods to support systematic reasoning about various quality attributes (e.g., real-time [6], reliability [7], and performance [8]) during software architecture design. However, these methods study a specific quality attribute in isolation. In reality, quality attributes interact with each other. For example, there is generally a conflict between configurability and performance [9]; performance also impacts modifiability, and each quality attribute impacts cost [10].

Some researches have developed methods to make quality attributes a central consideration during application design. Bosch [11] proposes a method that explicitly considers quality attributes during the design process. Hofmeister et al. [12] describe a framework known as global analysis to identify, accommodate, and describe architecturally significant factors including quality attributes early into the design phase. However, these methods do not sufficiently support the reasoning about the quality consequences of each design decision.

The work of Chung et al. [1] provides a framework that considers each design decision based on its effects on the quality attribute space. However, it does not provide support to explicitly perform trade-off analysis between competing design decisions.

Bass et al. have proposed the Attribute Driven Design (ADD) method [10] to help the architect base the design process on the desired quality attributes. ADD is basically built upon Attribute Based Architecture Styles (ABAS) [13], and architectural views [14, 15]. It provides a framework to make design decisions with known affects on the desired quality attributes. However, the codified knowledge or experience of an architect may present more than one design alternatives for each design decision. In this situation, a quantitative reasoning framework to support the multi criteria decision analysis can complement methods like ADD.

2.2. Quantitative reasoning for design decisions

Kazman et al. [4] propose the Cost Benefit Analysis Method (CBAM) to quantify design decisions in terms of cost benefit analysis. They apply a quantitative approach to evaluate and select from competing architectural strategies. The idea is that stakeholders will essentially choose the strategies that maximize their return on investment (ROI). CBAM, however, does not help identifying architectural candidates, since design decisions are considered independently. Rather, it relies on other methods like Architecture Tradeoff Analysis Method (ATAM) [16] to identify competing architecture strategies. ATAM is a SA evaluation method, which itself needs a SA as an input to the evaluation process.

Mikael et al. [3] developed a quantitative approach to support the comparison of candidate architectures using Analytical Hierarchy Process (AHP). This method provides a structured way of eliciting stakeholders' preferences for desired quality attributes and helping them gain quantified understanding of the benefits and liabilities of different architecture candidates.

The goal of [3] is similar to ours, but the approach differs in a number of ways. Like CBAM, this method assumes that a small set of architecture candidates have been created, but gives no guidance on how these alternatives are supposed to be reached. In contrast to existing approaches, which evaluate coarse-grained architecture candidates, our approach offers guidance quite early during the architectural design process. This is achieved through the evaluation of various fine-grained design options, which together produce the resulting SA.

3. Proposed Approach

3.1. Goals of the approach

Our goal is to reduce the complexity and increase the reliability of SA design. One way of doing it is by systemizing the architectural design process, as suggested earlier in [23]. This would help architects to systematically determine the optimal combination of design alternatives (i.e. the best candidate architecture). To achieve this goal, our approach must satisfy two main requirements:

1. **Local Requirement:** for each individual design decision, the approach should select the alternative that best matches stakeholders' preferences on associated quality attributes.
2. **Global Requirement:** the selection of one or a combination of individual design alternatives must not violate stated project constraints, and must also maintain decisions' inter-dependencies.

For the first requirement, we need to use a mechanism by which we can compute values offered by each alternative with respect to their corresponding design decisions. The higher the value for one alternative is, the better it becomes comparatively to others. Hence the values computed must be influenced by stakeholder preferences on quality attributes.

However, for a particular design decision, it may not always be the case that the alternative with the highest value obtained will be selected, as it may result in a constraint violation. For example, the selection of a particular alternative may incur additional costs on top of costs incurred by other decisions, violating the cost constraint stated by participating stakeholders. This example shows the global requirement that must be met.

The design approach must also deal with inter-dependencies among different decisions. Initially, we identified two types of dependencies (*Alternative-Based* and *Context-Based* dependencies), which we discuss in section 4.2. As an example, assume that we have two design decisions, each having several alternatives. If the best alternative (yielding the highest value score) for the first design decision conflicts with the best alternative from the second design decision, then it is difficult for the architect to resolve the clash.

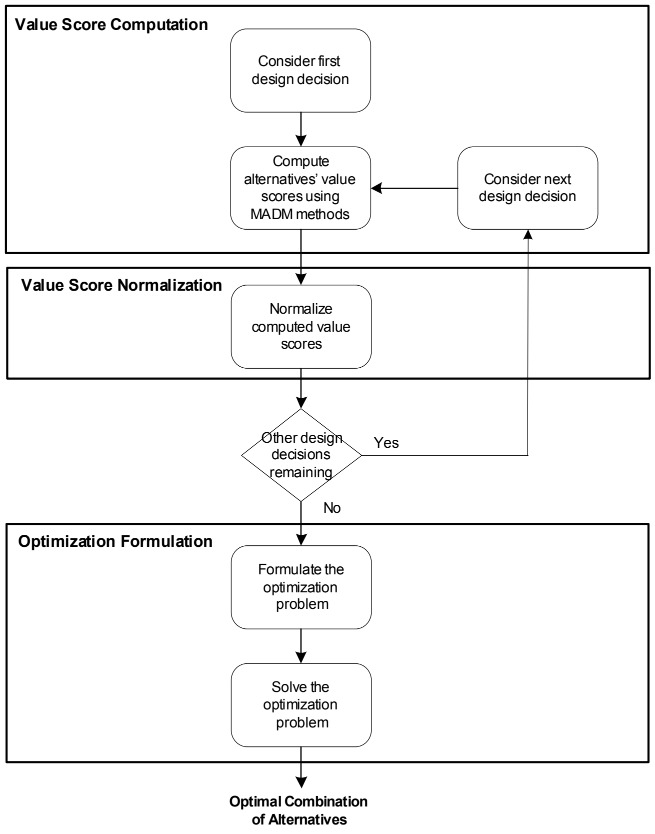
Based on these observations, we argue that the SA design problem can be formulated as a global optimization problem, since design decisions are highly dependent on each other, and the selection of any design alternative must not violate global constraints. Therefore, we state the optimization problem as follows:

"we seek to maximize the value of the SA for all stakeholders involved by selecting alternatives yielding highest value scores, whilst assuring that dependencies are maintained and global constraints stated are not violated".

To the best of our knowledge, none of the existing design approaches explore the potential of applying optimization techniques for solving complex software design problems, comprised of multiple inter-dependent architectural design decisions.

3.2. The ArchDesigner Approach

ArchDesigner comprises three steps, as shown in Figure 1. It starts with the first design decision and computes value scores for its potential alternatives solutions. This is repeated for each design decision. The second step transforms the computed value scores into a normalized form, in order to prepare them for the third step. Finally, the third step formulates the optimization equations so as to maximize the values associated with selected alternatives, subject to stated constraints and inter-dependencies.



Value Score Computation
Consider first design decision
Compute alternatives' value scores using MADM methods
Consider next design decision
Value Score Normalization
Normalize computed value scores
Other design decisions remaining
Yes
No
Optimization Formulation
Formulate the optimization problem
Solve the optimization problem
Optimal Combination of Alternatives

Figure 1. ArchDesigner process

3.2.1. Step 1: Value score computation

We define the *value score of a design alternative* as the degree to which an alternative satisfies the desired quality attributes. For a particular design decision, potential design alternatives are evaluated across a set of quality attributes associated with that design decision. Figure 2 depicts the process of computing alternative value scores pertaining to a particular design decision. The input to this process is twofold:

1. Design alternatives and their relative support for associated quality attributes. One alternative, for instance, may offer high reliability but low performance. Another one may provide high performance at the expense of security.

2. Preferences on associated quality attributes provided by different stakeholders. For example, one stakeholder may be more concerned with performance and loose-coupling at the expense of reliability and modifiability. Another stakeholder may be more interested in modifiability and implementation complexity only.

For the computation method, we rely on Multiple Attribute Decision Making (MADM) methods [17], which are widely used in different business areas. They enable stakeholders to make preference decisions about the available alternatives that are characterized by multiple, usually conflicting, attributes.

There are several MADM methods available, including AHP, SAW, and ELECTRE. In this paper, we employ the Analytic Hierarchy Process (AHP) method [18]. ArchDesigner is, however, flexible enough to accommodate other MADM methods, at the discretion of the different stakeholders.

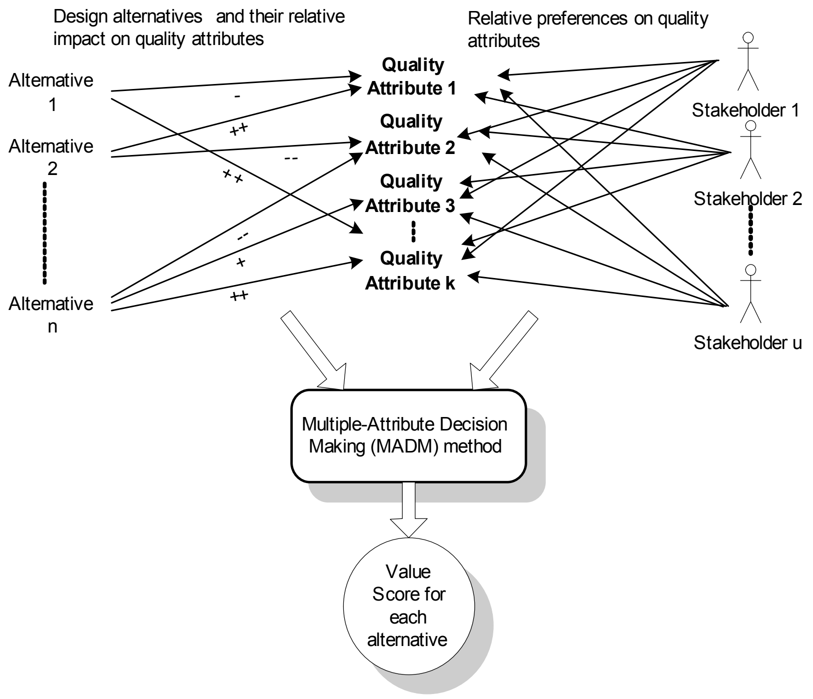

Design alternatives and their relative impact on quality attributes
Relative preferences on quality attributes
Alternative 1
Alternative 2
Alternative n
Quality Attribute 1
Quality Attribute 2
Quality Attribute 3
Quality Attribute k
-
++
--
++
--
+
++
Stakeholder 1
Stakeholder 2
Stakeholder u
Multiple-Attribute Decision Making (MADM) method
Value Score for each alternative

Figure 2. Computation of value scores

The AHP process

Unlike other MADM methods, AHP relies on relative weighting (pair-wise comparison) and is thus less sensitive to judgment errors common to other methods using absolute assignments. Basically, AHP comprises four main steps:

- *Preparation:* this step articulates the different elements involved in the decision-making process for the designated decision. It entails identifying design alternatives, quality attributes that will be used for evaluating these alternatives, and also the different stakeholders participating in this decision.
- *Weighting Quality Attributes:* the aim of this step is to determine the relative weight for every quality attribute. For each design decision, each stakeholder provides their preferences on relevant quality attributes by comparing every pair of quality attributes

(Q_a,Q_b), using AHP's weighting scale shown in Table 1. This is used to determine how important Q_a is, in comparison to Q_b. Note that values (2, 4, 6, and 8) represent compromises between these preferences. This means that for k quality attributes, $\frac{k(k-1)}{2}$ pair-wise comparisons will need to be made by every stakeholder. Stakeholder preferences on quality attributes are aggregated before we compute quality attributes' weights W_z, for $1 \geq z \geq k$.

- *Weighting Alternatives' Quality Support*: next we determine how each design alternative relatively supports the relevant quality attributes. For every quality attribute Q_a, we compare the n potential design alternatives in a pair-wise fashion. The values in Table 1 are used for assigning weights, to determine how alternative A_x supports quality attribute Q_a in comparison to alternative A_y. We can then derive the relative support each alternative is offering to every quality attribute. This yields us an n x k support matrix:

$$S = (S_{xa}; 1 <= x <= n, 1 <= a <= k)$$

where every entry (x,a) corresponds to how alternative A_x relatively supports quality attribute Q_a.

Table 1. AHP weighting scale

If A is … as (than) B	Quantitative Weight
equally important	1
moderately more important	3
strongly more important	5
very strongly more important	7
extremely more important	9

- *Computing Value Scores:* we can now compute the value score V_{ij} for alternative i of design decision j using the following formula:

$$V_{ij} = \sum_{z=1}^{k} W_z S_{iz} \qquad (1)$$

3.2.2. Step 2: Normalization of value scores

Before moving on to the third step, we need to normalize the alternatives value scores obtained in the previous step. The reason for this is that the alternatives value scores from different decisions will be summed in the next step. Hence, we have to scale them relatively. To do this, we weight the different decisions N_j relatively in a manner that reflects their relative significance to the application. Naturally, some design decisions are more important than others, and thus should receive higher weights. Having done this, we then multiply the obtained value scores by the weight of their corresponding decision:

$$V_{ij}^{'} = N_j \times V_{ij} \qquad (2)$$

3.2.3. Step 3: Optimization formulation

In this step, we seek to maximize the accumulative value score, which represents the *objective function*. This can best be formulated using Integer Programming (IP):

Maximize:

$$\sum_{j=1}^{m} \sum_{i=1}^{n_j} X_{ij} V'_{ij} \tag{3}$$

Subject to:

$$\forall j \in [1,...,m] : \sum_{i=1}^{n_j} X_{ij} = 1 \tag{4}$$

$$Cost(X_{i_1 1}, X_{i_2 2}, ..., X_{i_m m}) <= \text{Constraint}_{\cos t} \tag{5}$$

$$Time(X_{i_1 1}, X_{i_2 2}, ..., X_{i_m m}) <= \text{Constraint}_{time} \tag{6}$$

$$X_{ij} + X_{ab} <= 1 \quad , \quad j \neq b \tag{7}$$

Where:

$m \geq 1$ denotes the number of design decisions considered.

$n_j \geq 2$ denotes the number of alternatives within design decision j.

$X_{ij} \in [0,1]$, where 1 denotes that alternative i is selected for design decision j, and 0 denotes non-selection of this alternative.

V'_{ij} denotes the normalized value score for alternative i of design decision j.

$\text{Constraint}_{\cos t}$ denotes cost constraint.

Constraint_{time} denotes time constraint.

$Cost(X_{i_1 1}, X_{i_2 2}, ..., X_{i_m m})$ and $Time(X_{i_1 1}, X_{i_2 2}, ..., X_{i_m m})$ are functions that take a list of alternatives as an input and compute the expected cost and time respectively required to implement this combination.

Equation 3 maximizes the accumulative value score by selecting a combination of alternatives yielding the highest value score possible. Equation 4 guarantees that only one alternative is selected for each design decision. Equations 5 and 6 guarantee that the alternatives selected do not violate the cost and time constraints. Note that *Cost()* and *Time()* are generic functions that are influenced by the project context in which they are applied. These functions can, however, leverage generic estimation models, such as COCOMO [19], for estimating the cost and time required. Finally, equation 7 enforces dependencies among alternatives from different decisions, particularly the case where two alternatives cannot be selected at the same time.

4. Case Study

4.1. The Project

The Glass Box (GB) project [20] is a part of a multi-year, research program[1] to generate new tools and technologies for information analysts. The GB itself is a production software system, which is deployed in the analyst's working environment. Its basic role is to capture detailed information on a user's workstation activities during information gathering and analysis tasks. This information is recorded over long time periods, and made available to a range of research projects based at research labs across North America. There are approximately 15 separate research projects funded by the overall program.

The GB application is further intended as an integration and test platform for the participating research projects. The research projects are required to link their software into the GB environment, and demonstrate their capabilities in helping analysts to solve real problems. This requires instantaneous notification of the analyst's actions, such as opening a document or performing a search. Also, in order to share knowledge generated from each research tool, there must be mechanisms for storing data generated from each tool and notifying other tools of its existence.

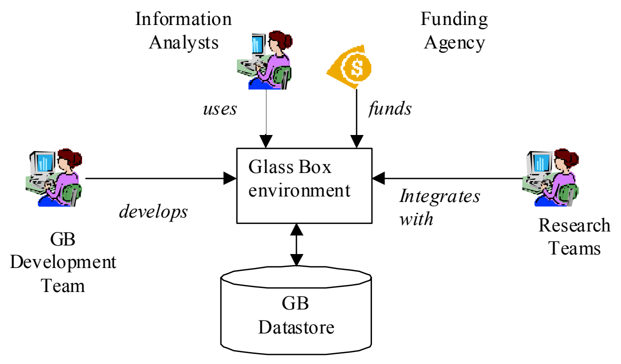


Figure 3. Glass box stakeholders

Figure 3 depicts the relationship between the GB application and the various stakeholders involved. *Information Analysts* are the primary users of the GB. It is integrated with their work environment, and provides both transparent and explicit tools to record actions. The *GB Development Team* is responsible for all aspects of GB design, development and deployment. The *Research Teams* see the GB as a software tool and data repository. They need to access the data in the repository, and programmatically integrate their tools with the GB environment. The *Funding Agency* is responsible for approving development plans and allocating associated budgets, and for the overall success of the program.

The GB project commenced in 2002, and in early 2003 the first version of the software was released and successfully deployed. This version was primarily concerned with satisfying the needs of the *Information Analysts* in capturing and storing their work activities.

[1] http://www.ic-arda.org/Novel_Intelligence/As

The initial GB version was a 2-tier client-server system, utilizing a database, file store, and a set of tools to capture user activities when they accessed Web sites, documents, and commenced and completed assignments. It ran standalone on each user workstations. Nightly scripts extracted the data from individual databases and merged them into a central data store for periodic distribution to the *Research Teams*.

Immediately after this release, planning for version 2 commenced. While the focus of version 1 had been the creation of an information capture environment for analysts, the next version had a much wider set of requirements. Briefly, these were:

- Provide programmatic access to the GB database for the research project teams.
- Provide a mechanism to immediately inform the research tools when an analyst performs an action that is of interest to their technology.
- Provide secure programmatic access to a GB environment from a remote site.
- Ensure that the GB environment could scale to support deployments with large numbers of users.

An additional requirement was that this new set of functionality must be delivered in a one-year period based on a fixed development budget.

4.2. Analyzing the case study

We applied ArchDesigner as a post-mortem analysis of the major architectural design decisions that were made during the GB design. We then compared the results obtained by ArchDesigner with the design decisions that were actually made.

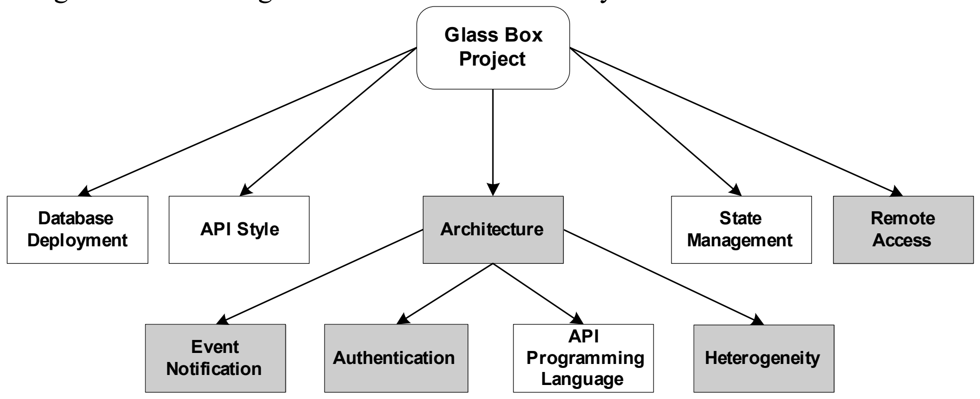


Figure 4. GB design decisions and their interdependencies

For this purpose, we held several interviews with the Lead Architect (LA) of the project. Initially, there were at least 9 architectural design decisions that had undergone extensive discussions and evaluation during the design stage. These decisions are shown in Figure 4. We have selected 5 decisions out of 9 to include in our study. These are represented with grayed boxes in Figure 4.

Directed arrows in Figure 4 depict inter-dependencies among the different decisions. For example, *State Management* and *Architecture* decisions are independent, while *Event Notification* and *Authentication* decisions are dependent on the *Architecture* decision. We use the terms *superior* and *inferior* to describe this kind of relationship. For example, the *Heterogeneity* decision is inferior to *Architecture* decision.

A list of the selected design decisions is shown in Table 2, along with the corresponding alternatives that were considered, relevant quality attributes, and the stakeholders participating in the decision-making process. Highlighted alternatives represent the real selections made.

Table 2. List of selected decisions, along with their alternatives, quality attributes, and stakeholders

Design Decision	Alternatives	Quality Attributes	Stakeholders
Architecture (ARCH)	• **3-tier using J2EE (THTJ)** • 3-tier using .Net (THTD) • 2-tier (TWOT) • COABS (COAB)	•Modifiability •Scalability •Performance •Cost •Development effort •Portability •Ease of installation	•Development team •Research Teams •Funding agency
Event Notification (EVNT)	•**Publish-Subscribe using JMS** (JMS) •Publish-Subscribe using MSMQ (MSMQ) •Database triggers (TRGR) •COABS (COAB)	•Reliability •Performance •Complexity of implementation	•Development team •Research Teams
Authentication (AUTH)	•**Database-based security (DB)** •J2EE-based security (J2EE) •.Net-based security (.NET) •COABS (COAB)	•Complexity of implementation •Ease of deployment and setup	•Development team •Research Teams
Remote Access (RMAC)	•Browser-based (HTTP) •**Web services (WEBS)** •Secure network (VPN)	•Performance •Security •Modifiability •Complexity of deployment •Complexity of implementation	•Development team •Research Teams •Funding agency
Supporting non-windows platforms for API (HETR)	•**Java Language (JAVA)** •Browser (BROW) •C Language (C)	•Usability •Modifiability •Cost •Development effort	•Research Teams •Development team

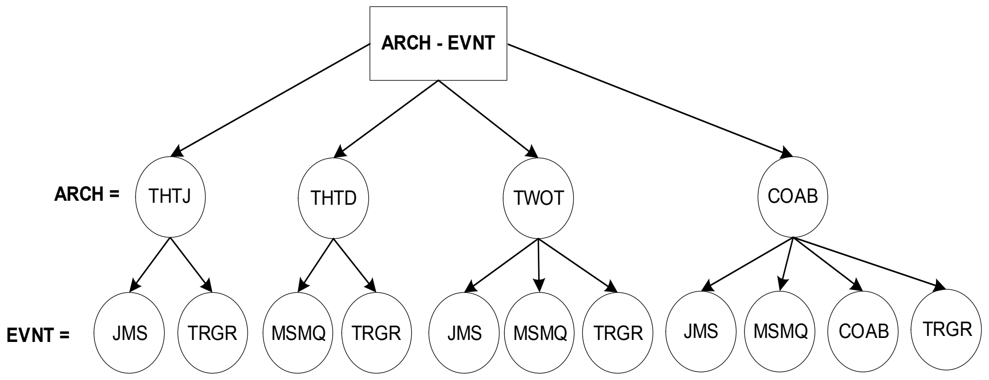


Figure 5. Dependencies between Architecture and Event Notification decisions

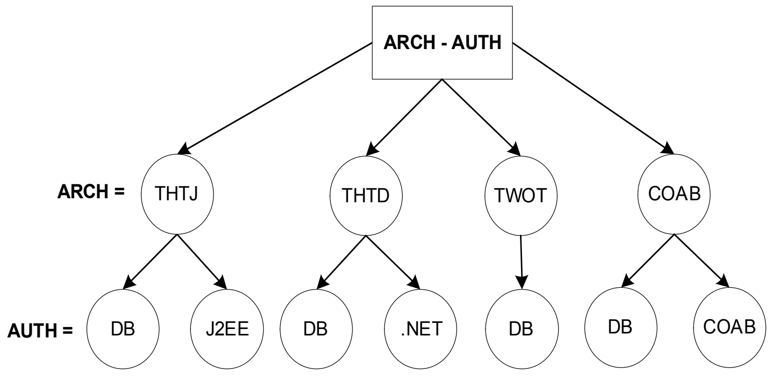


Figure 6. Dependencies between Architecture and Authentication decisions

To handle inter-dependencies inherent in the project, we identified two types of decisions inter-dependencies:

- *Alternative-Based Dependency*: Here the alternatives available in an inferior decision change depending on the alternative selected by its superior decision. Dependency examples are shown in Figure 5 and 6 respectively between the *Architecture* and *Event Notification* decisions and also between *Architecture* and *Authentication* decisions. In the latter, *Database-Based* (DB) is the only authentication alternative if the *Two-Tier* (TWOT) architecture alternative is selected.
- *Context-Based Dependency*: Here an alternative's support for certain quality attributes (for an inferior decision) may change positively or negatively depending on the alternatives selected by its superior decision. This we refer to as the *decision context*. As an example, consider the "Java" language alternative from the *Heterogeneity* decision. The dependency here stems from the fact that cost and development effort quality attributes will relatively drop (i.e. improves) if the *Three-Tier J2EE* (THTJ) architecture alternative is selected. This means that the same *Java* alternative will receive different value scores under different tree paths.

After analyzing the selected design decisions and their alternatives and inter-dependencies, 171 potential combinations of alternatives existed for the architect to consider. In addition, every design decision involved more than one group of stakeholders, each of which favored different solutions. Also, the project was constrained by a one year duration. All these issues made the design activity quite complex.

5. Application of the Approach

5.1. Value score computation

In this step, we leveraged a generic AHP tool, *ExpertChoice* [21], for computing alternatives' value scores. We applied AHP to every design decision in turn. For every decision, we asked the LA to provide us with the preferences on quality attributes that were expressed from the perspectives of different stakeholders. Due to space limitations, we show only how we computed value scores for the *Architecture* (ARCH) design decision.

Table 3 shows the quality attributes' preferences provided for the *Funding Agency* stakeholder. In several cases, certain quality attributes were not of any importance to a given stakeholder. The *Research Teams* stakeholder, for instance, was not concerned about the development costs incurred by the GB development team. We treated these cases by assigning them the lowest weight possible (1 or 9) in comparison to every other quality attribute.

Table 3. Preferences on quality attributes provided by Funding Agency stakeholder for ARCH decision

Stakeholder	1st Quality Attribute	2nd Quality Attribute	Quantitative Weight
Funding Agency	Modifiability	Scalability	5
	Modifiability	Performance	2
	Modifiability	Cost	1
	Modifiability	Dev. Effort	1
	Modifiability	Portability	3
	Modifiability	Ease of Inst.	1
	Scalability	Performance	1/4
	Scalability	Cost	1/5
	Scalability	Dev. Effort	1/5
	Scalability	Portability	1/2
	Scalability	Ease of Inst.	1/2
	Performance	Cost	1/5
	Performance	Dev. Effort	1/5
	Performance	Portability	2
	Performance	Ease of Inst.	1
	Cost	Dev. Effort	1
	Cost	Portability	5
	Cost	Ease of Inst.	4
	Dev. Effort	Portability	5
	Dev. Effort	Ease of Inst.	4
	Portability	Ease of Inst.	1/2

After gathering quality preferences for the stakeholders, we realized that these preferences varied from one stakeholder to another, resulting into different quality attribute priority ranks.

This is shown in Table 4. Different stakeholder preferences were then aggregated by computing the geometric mean for every quality attribute, resulting in the quality attributes' weights shown in the last column of Table 4.

Table 4. Quality attributes' weights computed for every stakeholder on ARCH decision

Quality Attributes	Stakeholders			*Aggregated*
	Dev. Team	**Research Teams**	**Funding Agency**	
Modifiability	0.216	0.294	0.184	0.280
Scalability	0.087	0.092	0.038	0.082
Performance	0.052	0.117	0.087	0.097
Cost	0.245	0.019	0.272	0.135
Dev. Effort	0.245	0.019	0.272	0.135
Portability	0.050	0.155	0.053	0.094
Ease of Inst.	0.106	0.304	0.093	0.177

We next asked the LA to provide us with each alternative's support for different quality attributes. This data was provided from the perspective of development team only, since it requires technical knowledge that only the development team possesses. Table 5 depicts the alternatives' pair-wise comparisons with respect to *portability* quality attribute.

Interestingly, some pair-wise comparisons were essentially impossible to weight - for example, those involving COABS [24] with respect to certain quality attributes such as performance and development effort. The LA was not able to determine the appropriate weight since the development team had limited exposure to the COABS technology, and no reliable benchmarks were available to compare this proprietary solution with other alternatives. We treated these comparisons by assigning them the weight 1, with the assumption that they were equally strong in supporting the questionable quality attributes.

Table 5. Pair-wise comparisons for ARCH alternatives with respect to portability attribute

Quality Attribute	1st Alternative	2nd Alternative	Quantitative Weight
Portability	THTJ	THTD	9
	THTJ	TWOT	9
	THTJ	COAB	1
	THTD	TWOT	1
	THTD	COAB	1/9
	TWOT	COAB	1/9

Using AHP, this data were transformed into the relative weights shown in Table 6. Using this table, we can tell how each alternative relatively supports every quality attribute. For example, *Three-Tier J2EE* seems to be the best alternative supporting *modifiability.*

Applying formula (1) on the results in Table 6 and aggregated weights shown in Table 4, we obtain the following value scores for ARCH decision shown in Table 7. Similarly, we computed value scores for all other decisions shown in Table 7. Note that the JAVA alternative for the *Heterogeneity* decision is duplicated, to address the context-based dependency with THTJ.

Table 6. ARCH Alternatives' support weights

Quality Attributes	Alternatives			
	THTJ	THTD	TWOT	COAB
Modifiability	0.521	0.182	0.046	0.250
Scalability	0.402	0.402	0.054	0.143
Performance	0.204	0.204	0.347	0.246
Cost	0.166	0.120	0.487	0.227
Dev. Effort	0.152	0.110	0.515	0.223
Portability	0.450	0.050	0.050	0.450
Ease of Inst.	0.168	0.368	0.256	0.208

5.2. Normalization

In this step, we asked the LA to weight the different decisions relatively so as to determine their relative significance to the GB application. Figure 7 plots the different decisions across a 10-point weighting scale.

We then multiplied the value scores obtained in the previous step by their corresponding decision's weights using formula (2), resulting in the normalized value scores shown in last column of Table 7.

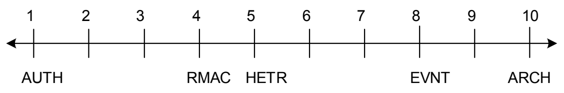


Figure 7. Weights of different design decisions

5.3. Optimization formulation

Before formulating the optimization problem, we need to determine the GB constraints and how each alternative impacts on these constraints. The GB version 2 was constrained by a one-year development schedule. Based on discussions with the LA, it seemed that it was hard (sometimes unrealistic) to exactly determine the time required to implement every particular alternative. However, it appeared that in some cases they were able to determine whether a particular alternative would violate the time constraint, without exactly specifying how long its implementation would take. This was the case with the *VPN Remote Access* and *J2EE-Based Authentication* alternatives, where it was believed that implementing either of these alternatives would exceed the schedule constraint.

Apart from these two alternatives, all other alternatives were feasible within the time constraint. To cope with this situation, we used the following formula:

$$Time(X_{i_1 1}, X_{i_2 2}, \ldots, X_{i_m m}) = \begin{cases} > 12 \text{ months} & \text{if } X_{34} = 1 \\ & \text{or } X_{23} = 1 \\ <= 12 \text{ months} & \text{otherwise} \end{cases}$$

The *Time()* function will return a number greater than 12 months only if the combination involves either X_{34} (*VPN Remote Access*) or X_{23} (*J2EE-Based Authentication*). X_{34} = 1 reads as the selection of 3rd alternative of 4th decision. This would result in a constraint violation according to formula (6).

Table 7. Obtained alternatives' value scores

Design Decision	Alternatives	Value Scores	Normalized Value Scores
Architecture (ARCH)	THTJ	0.314	3.14
	THTD	0.205	2.05
	TWOT	0.236	2.36
	COAB	0.246	2.46
Event Notification (EVNT)	JMS	0.258	2.064
	MSMQ	0.272	2.176
	TRGR	0.229	1.832
	COAB	0.241	1.928
Authentication (AUTH)	DB	0.215	0.215
	J2EE	0.358	0.358
	.NET	0.223	0.223
	COAB	0.204	0.204
Remote Access (RMAC)	HTTP	0.326	1.304
	WEBS	0.227	0.908
	VPN	0.446	1.784
Support non-windows platforms for API (HETR)	JAVA	0.257	1.285
	BROW	0.294	1.47
	C	0.155	0.775
	JAVA with THTJ	0.295	1.475

By applying the optimization equations, we obtained the following combination: *Three-Tier J2EE* for *Architecture*, *JMS* for *Event Notification*, *Database-Based* for *Authentication*, *Java Language* for *Heterogeneity*, and *Browser-Based* for *Remote Access.* This was recommended as the optimal candidate architecture with the highest accumulated value score of "8.198". This combination only varied in the case of the *Remote Access* decision from the alternatives that were actually chosen in the GB project (see Table 2).

6. Discussion

6.1. General findings

The following are the main findings of this study:

- The recommended candidate architecture shows an acceptable accuracy level for the proposed approach, since only one design decision (*RMAC*) was different from the actual design. RMAC was however the second lowest important decision.
- Overlooking context-based dependencies may lead to sub-optimal design decisions. This was the case with the *Heterogeneity* decision, as *Browser* alternative would have been

ranked first, if contextual influences from the superior decision (*Architecture*) were not considered.

- Based on feedback from the LA, the aggregation of stakeholders' preferences on quality attributes (for each design decision) seems to have produced a reasonable compromise between the stakeholders' conflicting quality goals.
- The project constraints have to be considered in the selection process. Time constraints excluded two alternatives from two different decisions (*Authentication* and *Remote Access*), though they were ranked first in their corresponding design decisions.
- Feedback on the approach from the GB LA was positive. The systematic and highly visible decision process would likely have been helpful to the GB team through the architecture design. It would have made the many discussions on design alternatives explicit, and aided communication within the team and justification of decisions to stakeholders.

6.2. ArchDesigner's contributions

The main contribution of ArchDesigner is the disciplined methodological support it offers to the SA design process. ArchDesigner does not claim to replace architects in making architectural decisions. It rather helps architects in:

- *Making informed architectural decisions*: Through quantitative evaluation of stakeholder quality preferences and design alternative support for quality attributes, ArchDesigner provides a formal way of rationalizing and justifying decisions. It also promotes a rigorous yet simple way of reasoning about architecture quality early in the design stage, through the explicit evaluation of quality attributes.
- *Engineering distributed SA systematically*: ArchDesigner offers a disciplined engineering approach for designing distributed software architectures. In a fine-grained fashion, it enables architects to examine and evaluate all candidate architectures and explore the tradeoffs between available design alternatives.
- *Alleviating design complexity*: ArchDesigner helps make the architecture design process more systematic by formally handling the various design decisions, alternatives, inter-dependencies, stakeholder quality goals, and project constraints. By managing all forces influencing the design process, it also helps determining the optimal combination of design alternatives.
- *Handling the concerns of different stakeholders*: Through an aggregation of stakeholder preferences on quality attributes, ArchDesigner strikes a balance among conflicting stakeholder quality goals. In addition, ArchDesigner facilitates the communication of architectural design decisions among different stakeholders, thus increasing the accuracy of decisions made.

6.3. Limitations

Some limitations were observed during the study:

- AHP's 9-point weighting scale is limiting, leading to inconsistencies. This was observed when comparing *Web Services* and *VPN* alternatives with respect to the *Complexity of*

Deployment quality attribute. We assigned this a weight of 9 as it was the highest weight possible, but we would have assigned it a relatively much higher weight if possible.

- Uncertainties in judgments were not formally treated, particularly judgments involving the *COABS* alternative. We believe that treating judgmental uncertainties would strengthen ArchDesigner in yielding more accurate results.
- By recalling the way the *Remote Access* decision was made, it appeared that a different cognitive approach was followed. Therefore, different decision-making approaches such as *profiles* [22] might be appropriate for certain design decisions.
- As reported in section 5.3, it was unrealistic to expect accurate estimates about the time required to implement every design alternative. However, it was possible to tell from experience whether a particular alternative would need more/less than 12 months to implement. A possible resolution is to have a relative weighting against the time constraint, which is called ideal-value rating [16].
- Judgment consistency level was not measured before computing value scores. Measuring the consistency level of stakeholders' judgments would help ensure the accuracy of the judgments. In case of inconsistent judgments, stakeholders may need to revise the weights.

7. Conclusion and Future work

In this paper, we proposed ArchDesigner as a systematic approach for facilitating the architectural design of distributed software applications. Using optimization techniques, it attempts to determine the optimal combination of design alternatives that best satisfy stakeholders' quality goals and project constraints. To provide validation of the approach, we applied it as a post-mortem analysis to an already implemented project, and the results showed a high level of accuracy for ArchDesigner.

As reported in the previous section, there were few limitations observed from the results obtained. We plan to address these limitations. Moreover, since our study was conducted as post-mortem, the potential for bias in the recollections of the LA exists, which may have influenced the AHP scoring. We also simplified the design problem by only selecting a subset of the design concerns in the GB project. Thus, we intend to conduct another experiment during the SA design stage of a system. We also plan to build a tool based on the ArchDesigner to support SA design process.

References

[1] L. Chung, et al., *Non-Functional Requirements in Software Engineering*: Kluwer Academic Publishers, Boston, Ma.,1999

[2] D. Anderson, D. Sweeny, and T. Williams, *An Introduction to Management Science: Quantitative Approaches to Decision Making*: South-Western Eductional Publishing,2002

[3] M. Svahnberg, C. Wholin, and L. Lundberg, *A Quality-Driven Decision-Support Method for Identifying Software Architecture Candidates.* Int. Journal of Software Engineering and Knowledge Engineering, 2003. **13**(5): p. 547-573.

[4] R. Kazman, J. Asundi, and M. Klein. *Quantifying the Costs and Benefits of Architectural Decisions*. *Proc. of the 23rd International Conference on Software Engineering*. 2001. Toronto, Canada.

[5] *IEEE Standard 1061-1992, Standard for Software Quality Metrics Methodology*. New York: Institute of Electrical and Electronic Engineers,1992

[6] M.H. Klein, et al., *A Practitioner's Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems*: Kluwer Academic,1993

[7] M.R. Lyu, ed. *Handbook of Software Reliability Engineering*. 1996, McGraw-Hill and IEEE Computer Society: New York.

[8] C.U. Smith and L.G. Williams, *Software Performance Engineering: A Case Study Including Performance Comparison with Design Alternatives.* IEEE Transactions on Software Engineering, 1993. **19**(7).

[9] L. Lundberg, et al. *Quality Attributes in Software Architecture Design*. *Proceedings of the IASTED 3rd International Conference on Software Engineering and Applications*. Oct., 1999.

[10] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*. 2 ed: Addison-Wesley,2003

[11] J. Bosch, *Design & Use of Software Architectures: Adopting and evolving a product-line approach*: Addison-Wesley,2000

[12] C. Hofmeister, R.L. Nord, and D. Soni, *Applied Software Architecture*. Reading, MA: Adison-Wesley Longman,2000

[13] M. Klein and R. Kazman, *Attribute-Based Architectural Styles*, Tech. Report, CMU/SEI-99-TR-022, Soft Engineering Institute, Carnegie Mellon University

[14] P.B. Kruchten, *The 4+1 View Model of architecture.* Software, IEEE, 1995. **12**(6): p. 42-50.

[15] D. Soni, R.L. Nord, and C. Hofmeister. *Software Architecture in Industrial Applications*. *Proc. of the 17th International Conference on Software Engineering*. 1995. Washington, U.S.A.: ACM Press.

[16] P. Clements, R. Kazman, and M. Klein, *Evaluating Software Architectures: Methods and Case Studies*: Addison-Wesley,2002

[17] K.P. Yoon and C. Hwang, *Multiple Attribute Decision Making: An Introduction*: Sage Publications,1995

[18] T.L. Saaty, *The Analytica Hierarchical Process*: McGraw-Hill,1980

[19] B. Boehm, *Software Engineering Economics*: Prentice-Hall,1981

[20] I. Gorton and J. Haack. *Architecting in the Face of Uncertainty: An Experience Report*. *Proc. International Conference on Software Engineering*. 2004. Edinburgh, Scotland.

[21] Expertchoice, http://www.expertchoice.com Last accessed 20th Aug., 2004

[22] M. Morisio, I. Stamelos, and A. Tsoukias. *A New Method to Evaluate Software Artifacts against Predefined Profiles*. *Proc. of Workshop on Software Enginering Decision Support Methodologies: SEKE*. 2002.

[23] T. Al-Naeem, F. Rabhi, B. Benatallah and P. Ray, *Towards Systematic Approaches for Designing B2B Applications*, Int'l Journal of Electronic Commerce (IJEC), 2004, to appear

[24] COABS, http://coabs.globalinfotek.com Last accessed 20th Aug., 2004