Automating Architecture Trade-Off Decision Making through a Complex Multi-attribute Decision Process

Majid Makki, Ebrahim Bagheri, and Ali A. Ghorbani

Faculty of Computer Science,
University of New Brunswick, Fredericton, Canada
{majid.makki,e.bagheri,ghorbani}@unb.ca

Abstract. A typical software architecture design process requires the architects to make various trade-off architecture decisions. The architects need to consider different possibilities and combinations of tactics and patterns to satisfy the elicited quality scenarios of the intended software system, some of which may be conflicting or inconsistent in nature. The formation of the correct composition of these elements of architecture decisions for the satisfaction of the quality scenarios can be considered an important art of the architect; however, in cases where the architect is dealing with multiple stakeholders with inconsistent preferences, this can be an awkward task. In this paper, we formalize this process as a complex multi-attribute decision making procedure within the Attribute Driven Design methodology. In such a context, we are able to incrementally elicit the communal preferences of the stakeholders with regards to the available quality scenarios and hence assist the software architect in methodically making the architecture decisions with the highest expected utility for the stakeholders.We will also introduce our implementation of a decision support system, which embodies the methods proposed in this paper, along with a case study.

1 Introduction

It is commonly accepted that quality and functional requirements are orthogonal in nature. The functional requirements of a software system could be achieved even without considering the notion of software architecture; therefore, the main focus of software architecture is on the satisfaction of quality requirements such as modifiability, availability, and performance. In addition, quality attributes of a software system need to be considered collectively and an isolated analysis of those attributes may not result in a comprehensive solution. For instance, Bass et al point out that the full satisfaction of security and availability measures is not simultaneously achievable. Better stated, active redundancy is a reasonable architecture tactic for elevating the level of availability of a software system; however, this may increase the vulnerability of the system [2].

The Attribute Driven Design (ADD) method is a recursive decomposition process aiming to address the aforementioned issue where at each stage a subset

R. Morrison, D. Balasubramaniam, and K. Falkner (Eds.): ECSA 2008, LNCS 5292, pp. 264–272, 2008.

of the quality scenarios that need to be satisfied are chosen and some architecture decisions, in terms of applying appropriate tactics or patterns, are made [3]. On the other hand, the importance of acquisition processes and management of uncertainties associated with the requirements and technical solutions have been previously mentioned in [4]. Even though, the formality of the ADD method assists the architects in effectively designing the architecture, the success of the decision making process involved in each iteration of ADD is highly dependent on the ability of the architect to interact with and elicit stakeholders' preferences over the quality scenarios.

In their recent critical analysis of the software architecture domain, Shaw and Clements have put forth several intriguing ideas that need to be addressed [10]. There, the development of practical and sophisticated automated architecture design assistants that can aid the architects in exploring the relationship between architectural design decisions and quality attributes is considered to be important. In an early attempt, a rule-based expert system called ArchE has been developed at SEI-CMU that serves as a software architecture design assistant by incorporating a body of knowledge about how quality requirements can be achieved using different architecture patterns and tactics [1]. The required input of this tool is a set of quality requirements (limited to performance and modifiability at this stage) and the output is the architecture design for the given requirements.

Experience shows that different stakeholders may prefer dissimilar quality requirements [2], which will impose various design constraints that can be inconsistent or conflicting in practice. In this paper, we take a more stakeholder-centric approach to architecture trade-off decision making in which the conflicting stakeholders' preferences over the quality requirements are incrementally elicited through a utility elicitation procedure. The architecture decision with the highest expected utility (based on the elicited utilities of the outcome of each architecture decision for the stakeholders) would be suggested as an optimal architecture decision in each round.

2 ADD+: An Extension to ADD

In the ADD method, it is assumed that all of the quality requirements of the system have been elicited and in each round, the architect is responsible for choosing a subset of the concrete quality scenarios that need to be satisfied at that stage. An architect may struggle with two main issues in this approach:

1. Stakeholders may have dissimilar preferences over quality scenarios. For example, the maintenance organization stakeholder usually prefers modifiability quality scenarios over performance, whereas the end user stakeholder prefers the vice-versa. One simplistic solution is to elicit the stakeholders' ordinal preferences with regards to the quality scenarios where quality scenarios of higher order are preferred over the others. In this approach, one may infer that quality scenarios of lower order should be sacrificed for the satisfaction of higher order quality scenarios when conflicts arise. However

while in most cases, a decision considering a trade-off between the conflicting quality scenarios results in a better outcome, the decision about the optimal trade-off is dependent upon the architect's expertise.

2. Although software architecture plays the most important role in achieving the quality requirements, architecture decisions do not always guarantee the promised level of quality for the software with regards to the quality scenarios of that iteration. This is due to various factors such as later decisions made in detailed design, implementation and deployment issues, and probable runtime uncertainties [2,5,9]. Such kind of uncertainty does not allow the architect to be fully confident of the outcomes of his/her decisions while interacting with the stakeholders in the preference elicitation process.

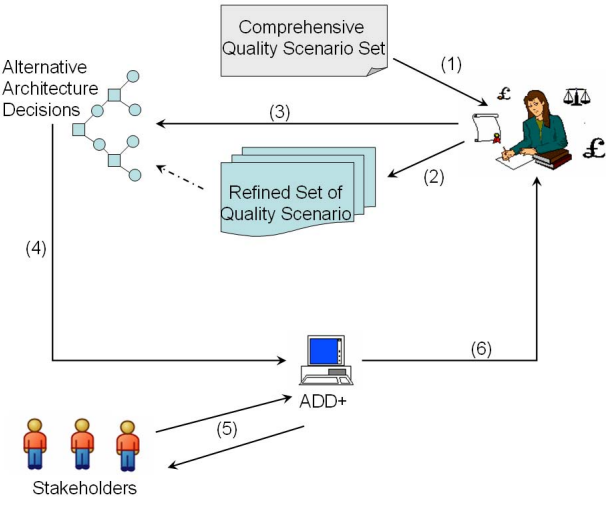

Fig. 1. The Workflow of ADD+

In this paper, we propose an extension to the ADD method (ADD+) by incorporating the role of system stakeholders in making architecture trade-off decisions. At each stage of ADD+, the architect chooses a subset of concrete quality scenarios, and specifies those quality scenarios within this subset that are conflicting. Based on the conflicting scenarios, the architect is required to propose various alternative architecture decisions along with the degree of quality scenario satisfaction obtained as a result of that architecture decision. A realistic approach should not make oversimplifying assumptions with regards to the degree of satisfaction of each quality scenario due to the uncertainties imposed by the before-mentioned factors. Once the set of alternative architecture decisions and their probable outcomes are specified by the architect, the preference elicitation process of the ADD+ commences: in each round of preference elicitation, system stakeholders are asked to answer a standard gamble query. Based on the stakeholders' responses, the utility of the outcomes of each architecture decision can be deduced. The architect can iterate over the steps of preference elicitation

until he/she is satisfied with the gained knowledge about the stakeholders' preferences. This information would allow the architect to make the optimal choice with regards to the most suitable architecture decision. Figure 1 summarizes this process.

3 Formalizing the Decision Problem

It is possible to model the process of stakeholders' preference elicitation and architecture decision making as a multi-attribute decision problem [6,7]. Here, the multi-attribute decision problem consists of attributes which represent the degree of satisfaction of the conflicting quality scenarios and the outcome is the quality of the resulting software system marginalized to those quality scenarios.

Let $CQS^i = \{cqs_1^i, ..., cqs_n^i\}$ be the set of conflicting quality scenarios at the i^{th} iteration of ADD+, $R^i = \{r_1^i, ..., r_n^i\}$ be their corresponding response measures and $AD^i = \{ad_1^i, ..., ad_k^i\}$ be the set of alternative architecture decisions at that iteration. Furthermore, suppose r_j^i is a random variable, and $\forall ad_l^i \in AD^i$, ad_l^i satisfies cqs_j^i to the degree of r_j^i according to a Gaussian probability distribution $P_{l,j}^i \sim N(\mu_{l,j}^i, \sigma_{l,j}^i)$[1].

Since the utility elicitation and decision making methods adopted here [6,7] assume that the set of outcomes is finite and countable, a discrete approximation of the Gaussian probability distribution is needed. The Pearson-Tukey approximation provides a three point representation of the Gaussian distribution. The representative points are the 5%, 50%, and 95% points and the probability of each point is 0.185, 0.63, and 0.185, respectively [8]. For a response measure r_j^i and an architecture decision ad_l^i, these three points are $r_{l,j_1}^i = -0.645 \times \sigma_{l,j}^i + \mu_{l,j}^i$, $r_{l,j_2}^i = \mu_{l,j}^i$, and $r_{l,j_3}^i = 0.645 \times \sigma_{l,j}^i + \mu_{l,j}^i$.

Suppose that we define a snapshot of the software system as a n-tuple s, which represents the state of the system at a certain time with regards to R^i. The set of all snapshots of the system, denoted S^i, consists of all possible combinations of configurations of R^i according to AD^i. S^i contains $k \times 3^n$ snapshots. The aim of the decision making problem is to choose the architecture decision whose corresponding snapshots (along with the probabilities of occurrence of those snapshots) satisfy the stakeholders' preferences and interests the most. So the set of attributes in this problem is R^i, the set of alternative decisions is AD^i, and the set of feasible outcomes is S^i.

4 Stakeholders' Preference Elicitation

The aim of preference elicitation is to elicit the utility of the snapshots for the stakeholders so that decision making can be done more confidently. The domain of a utility function u is S^i and its range is the real interval [0,1]. We do not need to know the complete utility function in order to make the optimal decision.

[1] Any other probability distribution could alternatively be used if a discrete approximation of it exists.

Instead we represent the utility of each snapshot by its lower and upper bounds. An elicitation query asks the stakeholders whether the utility of a snapshot is greater or less than a given value. This type of question can be easily translated into a standard gamble query. The query reduces the uncertainty in the calculation of the expected utility of each decision and thus helps the architect in confidently making the optimal decision.

According to [6], the number of possible elicitation queries, in each round of elicitation, is equal to the number of intersection points. An intersection point for an outcome (here, the outcomes of a decision are snapshots associated with it) is a point in its utility interval that is decisive for optimal decision making i.e. the true utility of the outcome being greater or less than the intersection point changes the optimal decision. The main difference between the original version of utility elicitation method and the one adopted here is that we are dealing with more than one stakeholder. We skip the details of calculating intersection points since it is independent of the number of stakeholders. For a more detailed discussion on how to calculate intersection points see [6] .

The optimal utility elicitation strategy is the one which asks a query with the highest expected value. The expected value of a query is the expected amount of increase in the expected utility of the optimal decision after asking that query.

The expected utility of the optimal decision at the current stage (before asking the query) can be calculated using the following equation:

$$EUOCD = \sum_{s_i} p_{s_i} \times \frac{u^{\uparrow}_{s_i} + u^{\downarrow}_{s_i}}{2} \tag{1}$$

where s_i is an obtained snapshot after making the optimal decision, p_{s_i} is the probability associated with it, $u^{\uparrow}_{s_i}$ and $u^{\downarrow}_{s_i}$ are the upper and lower bounds of its utility, respectively.

In order to calculate the expected utility after asking the query, all possible stakeholders' responses to that query need to be considered. Note that each individual stakeholder has two possible responses to a specific query, thus the number of possible communal responses to a query is equal to the number of all stakeholders plus one. The probability of a communal response can be calculated using the following equation:

$$p_{cr} = \binom{\eta}{\eta_{>}} \times \Pi^{\eta}_{i=1} p_i \tag{2}$$

where η is the number of stakeholders, $\eta_{>}$ is the number of stakeholders who stated that utility of the corresponding snapshot is greater than the intersection point, and p_i is the probability of an individual response. The probability of an individual response is either $\frac{u^{\uparrow}_{s_i} - ip}{u^{\uparrow}_{s_i} - u^{\downarrow}_{s_i}}$ or $\frac{ip - u^{\downarrow}_{s_i}}{u^{\uparrow}_{s_i} - u^{\downarrow}_{s_i}}$ where ip is the intersection point at which the query is being asked.

After receiving the stakeholders' responses, the utility of the snapshot for which the query had been asked can be updated by two alternative approaches.

One approach is to perform a voting process and update the utility interval of the snapshot based on the majority vote (e.g. if most of the stakeholders stated that the utility of the snapshot is greater than the intersection point, lower bound of the utility interval would be updated to the intersection point). However, this approach ignores the minority vote. A more sound approach is to alter both boundaries of the utility interval based on the number of votes given to each response. In this approach, we update the utility boundaries according to the following equations:

$$u_{s_i}^{\downarrow} \longleftarrow ip - \frac{\eta_{<} \times (ip - u_{s_i}^{\downarrow})}{\eta} \tag{3}$$

$$u_{s_i}^{\uparrow} \longleftarrow ip + \frac{\eta_{>} \times (ip - u_{s_i}^{\uparrow})}{\eta} \tag{4}$$

Note that introducing weights to represent stakeholder importance is a trivial task e.g. in our implementation one can define three End User stakeholders and only one Marketing stakeholder to show that the End User stakeholder is three time more important than the Marketing stakeholder. As mentioned above, the expected value of a query is equal to the difference between the expected utility of the optimal decision after the query and the expected utility before asking the query (see Equation 1). The expected utility after a query, can be calculated by the multiplication of the probability of each communal response (Equation 2) with the expected utility after receiving the specific response. In order to calculate the expected utility after receiving the responses, we can update the boundaries according to Equations 3, 4 and the expected utility of the optimal decision is recalculated. Therefore, after calculating the expected value of each query, we can choose the query with the highest expected value. The round of elicitation would be stopped either when there is no more query with expected value greater than zero or when the expected value of the best query does not satisfy the architect.

5 Case Study

Based on these theoretical foundations, a decision support system has been implemented, which assists the architect in eliciting the stakeholders' preferences and making the optimal decision with the highest expected utility. In this section, we will go through a case study using the implemented system.

We customize the case of the Garage Door example from [2] where three different stakeholders are involved: the End User Stakeholder, the Marketing Stakeholder and the Maintenance Organization Stakeholder. Two high prioritized quality scenarios are defined: a modifiability quality scenario called Low Recovery Time After Failure and a business quality scenario called Low Price. Obviously, the Marketing Stakeholder prefers the latter and the Maintenance Organization Stakeholder prefers the former. However, the End User Stakeholder prefers a low price system that can be fixed in a reasonable amount of time in case of failure.

Table 1. Response Measures of the Case Study

	Low Price ($200)	*Low Recovery Time*(20 *minutes*)
Monolithic Design	N(90,2)	N(60,10)
Structured Design	N(75,5)	N(85,4)

At the first stage, the architect needs to know whether he should design a monolithic system or a well-structured system. Assume that performance quality scenarios of this project would not be affected by applying modifiability tactics such as *maintaining semantic coherence*, *generalizing the module*, *information hiding*, *maintaining existing interfaces* and *using an intermediary*. However, these tactics may have a significant effect on the price of the system i.e. one of the business quality scenarios. This is due to the time required to implement these tactics and moreover the experienced programmers needed to implement them. The most important issue the architect has to deal with is which architecture decision would satisfy the community of stakeholders the most. We will see how the decision support system implemented for ADD+ can assist the architect in this case.

In the system, the architect may define the set of quality scenarios. The response measures of Low Price and Low Recovery Time are defined as *$200* and *20 minutes*, respectively by the system requirement specifications. After starting the first iteration, the architect would choose these two quality scenarios as the set of conflicting scenarios for this iteration. Afterwards, alternative decisions could be defined in the manner shown in Figure 2. Table 1 shows the normal distributions over response measures of conflicting scenarios in this case. One of the decisions, which is called Monolithic Design, satisfies the Low Price quality scenario to a higher degree ($\mu = 90$) more confidently ($\sigma = 2$) and the Low Recovery Time scenario to a lower degree ($\mu = 60$) with more uncertainty ($\sigma = 10$).

After starting the rounds of elicitation, one of the influential queries posed to the stakeholders could be interpreted as the following:

Which option would you prefer?

1. *A system which costs $220 and can be fixed in 28 minutes after a failure.*
2. *A system which costs the lowest possible amount and can be fixed in the lowest possible amount of time with 90% chance or a system which costs the highest amount and can be fixed in the highest amount of time with 10% chance.*

Note that the first choice in the above query is the interpretation of one of the snapshots generated by the support system automatically.

After some rounds of elicitation, the expected value of the next query was 0.09 and the expected utility of the Monolithic Design was 0.68 and the expected utility of the Structured Design was 0.71. We stopped the elicitation process due to the low expected value of the next query. Note that the expected utility

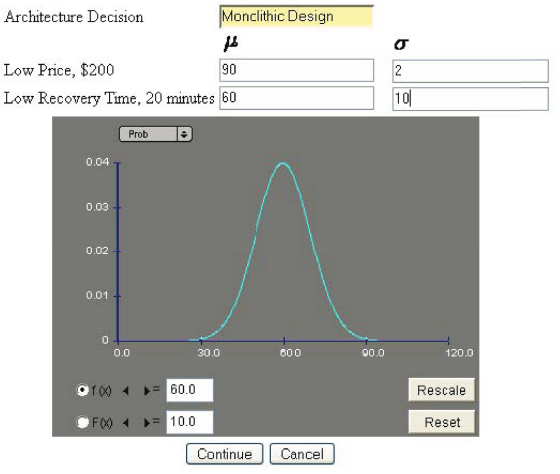

Fig. 2. Defining an Architecture Decision in the Decision Support System for ADD+

of alternative decisions are not accurate, but a rational decision could be made because the increase in the expected utility would not be greater than 0.09.

Since the end users are not satisfied either with a low price system with an extremely high recovery time or with an expensive system, which is recoverable in a reasonable amount of time, even the Marketing Stakeholder is not in favor of sacrificing the modifiability quality scenario for the sake of a lower price. As a result, the suggestion of the system, being the selection of the Structured Design alternative, makes sense.

6 Concluding Remarks

In this paper, we have reported a preliminary investigation of the applicability of decision theory and utility elicitation techniques for addressing the issue of stakeholders' preference elicitation by introducing an extension to the ADD method. Even though, we are at the early stages of our investigation, we believe that techniques employed here and some other similar techniques would be beneficial to the domain of software architecture. Many sources of uncertainty reported to be associated with software architecture decisions can be managed by theoretical foundations provided by decision theory and be automated using AI techniques based on those theories.

As future work, we are interested in both theoretical and empirical exploration of the relationship between the complexity of the conflicting quality scenarios and the number of elicitation queries posed to the stakeholders. We are also considering the evaluation of the proposed ADD+ method and its supporting toolset through various real-world case studies.

Acknowledgements

The authors graciously acknowledge the funding through grant RGPN 227441 from the National Science and Engineering Research Council of Canada (NSERC) to Dr. Ghorbani.

References

1. Bachmann, F., Bass, L., Klein, M.: Preliminary design of arche: A software architecture design assistant. Tech. rep., Software Engineering Institute Carnegie Mellon (2003)
2. Bass, L., Clements, P., Kazman, R.: Software architecture in practice. Addison-Wesley Longman Publishing Co., Boston (1998)
3. Bass, L.J., Klein, M., Bachmann, F.: Quality attribute design primitives and the attribute driven design method. In: van der Linden, F.J. (ed.) PFE 2002. LNCS, vol. 2290, pp. 169–186. Springer, Heidelberg (2002)
4. Brown, A.W., McDermid, J.A.: The Art and Science of Software Architecture. In: Oquendo, F. (ed.) ECSA 2007. LNCS, vol. 4758, pp. 237–256. Springer, Heidelberg (2007)
5. Celiku, O., Garlan, D., Schmerl, B.: Augmenting architectural modeling to cope with uncertainty. In: Proceedings of the International Workshop on Living with Uncertainties (IWLU 2007), co-located with the 22nd International Conference on Automated Software Engineering (ASE 2007), Atlanta, GA, USA, November 5 (2007), http://godzilla.cs.toronto.edu/IWLU/program.html
6. Chajewska, U., Koller, D., Parr, R.: Making rational decisions using adaptive utility elicitation. In: Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence, pp. 363–369. AAAI Press / The MIT Press (2000)
7. Fishburn, P.C.: Utility theory for decision-making. Wiley, New York (1970)
8. Pearson, E.S., Tukey, J.W.: Approximate means and standard deviations based on distances between percentage points of frequency curves. Biometrika 52(3-4), 533–546 (1965)
9. Poladian, V., Shaw, M., Garlan, D.: Modeling uncertainty of predictive inputs in anticipatory dynamic configuration. In: Proceedings of the International Workshop on Living with Uncertainties (IWLU 2007), co-located with the 22nd International Conference on Automated Software Engineering (ASE 2007), Atlanta, GA, USA, November 5 (2007), http://godzilla.cs.toronto.edu/IWLU/program.html
10. Shaw, M., Clements, P.C.: The golden age of software architecture. IEEE Software 23(2), 31–39 (2006)