An Integrated Tool for Trade-off Analysis of Quality-of-Service Attributes[1]

Leo Hatvani
Mälardalen Real-Time Research Centre,
Mälardalen University
Västerås, Sweden
leo.hatvani@mdh.se

Anton Jansen
ABB Corporate Research
Västerås, Sweden
anton.jansen@se.abb.com

Cristina Seceleanu
Mälardalen Real-Time Research Centre,
Mälardalen University
Västerås, Sweden
cristina.seceleanu@mdh.se

Paul Pettersson
Mälardalen Real-Time Research Centre,
Mälardalen University
Västerås, Sweden
paul.pettersson@mdh.se

ABSTRACT

In this paper, we present a tool for performing trade-off analysis of Quality-of-Service attributes of design solutions resulted from architectural, behavioral, or deployment changes in service-oriented systems. The tool allows for comparing the performance, reliability, and maintainability of such solutions, in an attempt to compute the optimal one with respect to the weighted sum of the considered quality attributes. Our tool uses the Analytic Hierarchy Process for computing these trade-offs and is integrated into the Quality Impact Prediction for Evolving Service-Oriented Software IDE. Consequently, architects and system analysts now have an easy to use tool set for making trade-offs for these system qualities.

Keywords

Quality-of-Service attributes, trade-off analysis, Analytic Hierarchy Process

1. INTRODUCTION

In Service-Oriented Systems (SOS), an essential factor when choosing a service out of functionally similar ones is the Quality-of-Service (QoS) that a specific service offers. Since any non-trivial modification made to the service architecture model inevitably influences several quality aspects, a central problem is to identify the effect of possible changes. By performing a systematic analysis of the possible trade-offs between QoS attributes,the risk of selecting a design solution with negative impact on important quality attributes can be reduced.

In this paper, we present a tool for the systematic trade-off analysis of QoS of SOS design solutions, which facilitates the comparison of *performance*, *reliability*, and *maintainability*. Our tool uses the *Analytic Hierarchy Process* (AHP) [7] for computing trade-offs, and is integrated into the Q-ImPrESS IDE, a Quality Impact Prediction for Evolving Service-Oriented Software framework. The novelty of our approach, as compared to the work of Zhu et al. [10], is integrating AHP with different automated analysis methods of the system's architecture and behavior.

In our approach, a small set of design solutions is assumed (less than 10, as the number of pair-wise comparisons in AHP method grows exponentially with number of design solutions). The quality of a design is equated with the weighted sum over the QoS attributes. Our tool, using the AHP method, determines the corresponding weight of each attribute metric. To achieve this, the tool uses the designer's experience and preferences. In the end, the usage of the tool results in a recommendation of which design solution to use in the design.

The rest of this paper is organized as follows: Section 2 presents the generic AHP method, the next section presents the adaptation we made to fit the method into the context of the Q-ImPreSS IDE, Section 4 describes the usage of the tool, whereas Section 5 presents the design decisions underlying the tool, lessons learned are covered in Section 6, and related work is presented in Section 7. Finally, the paper concludes in Section 8.

2. AHP

The Analytic Hierachy Process (AHP) [7] is a generic approach for multi-criteria decision making. Computing architectural trade-offs can be seen as a special case of such decision making, in which different criteria are represented by the functional and quality requirements of the system. The decision making targets the selection of the architectural solution that suits best these criteria.

The AHP method consists of 4 steps: (1) create a decision hierarchy, (2) determine relative criteria, (3) assess alternatives, and (4) interpret results. The first step identifies the criteria and associated analysis results to be used in the decision making. In addition, the alternatives to be considered should be decided upon. The second step consists of pair-wise comparisons of the selected criteria, to establish the importance of each criterion. In this pair-wise comparison, two criteria are put on a weight scale and the architect has to indicate *how much* one of the criteria is preferred over the other, using a predefined scale. In step three, the alternatives are pair-wise compared for each criterion. For each comparison, the relevant analysis results are presented alongside a weight scale similar

[1]This work was supported by the European Union under the ICT priority of the 7th Research Framework Programme in the context of the Q-ImPrESS research project.

to the one used in the previous step. In the fourth and last step, the architect examines the resulting scores of the alternatives with each other. To improve understanding, we show the contribution of each criterion to the end result.

3. AHP METHOD WITHIN Q-IMPRESS IDE

The Q-ImPrESS IDE [2] integrates various tools for the development of SOS. For example, it includes reverse engineering tools like SISSy and SoMoX [3], Palladio-based[4] performance prediction tools, Klaper-based[5] reliability predicton tools and KAMP-based [8] maintainability prediction tools. In the IDE, multiple design alternatives for SOS can be created and their quality with respect to various QoS attributes can be predicted. Our tool inside this IDE implements the Analytic Hierarchy Process method for evaluating trade-offs between these quality prediction results with the goal to reach a decision on which alternative should be pursued in futher development.

The Q-ImPrESS IDE operates on a limited number of QoS attribute metrics: Response Time/Throughput, Reliability, Utilization, and Cost/Effort. The AHP method is implemented using the wizard paradigm, so our tool is named "AHP Wizard". The AHP Wizard automatically pulls the required data from the Q-ImPrESS IDE, and asks the user to enter only the pairwise comparisons between criteria, and pairwise evaluations of values for these criteria, respectively. To simplify assessing the alternatives, we have made significant customization of the AHP method for Q-ImPrESS IDE. In this way, the tool is usable for users without AHP knowledge. The main customization is that the AHP decision hierarchy is predefined in the tool. We have found that, with a limited and constant number of qualities, a one level decision hierarchy is the most suitable for the tool's purpose. Data that is used by the user is automatically pulled from the results repository of the Q-ImPrESS IDE and aggregated into overview values. The comparisons that are integral to AHP Wizard are realized by a series of option buttons. Each of the option buttons is assigned to one of the following values: (i) Extremely preferred, (ii) Very strongly preferred, (iii) Strongly preferred, (iv) Moderately preferred, (v) Equally preferred. The options are displayed twice to make the display of comparisons more compact, and to logically disable inputting conflicting data.

4. THE USAGE OF AHP WIZARD

To access the AHP Wizard, the user has to mark the alternative designs that are going to be considered for the trade-off analysis. To make this selection as easy as possible and to maintain a consistent work-flow, we have integrated the invocation of the AHP Wizard into the Result Viewer, which is the tool used to display the results of the already run quality predictions. This tool and its interface can be seen in Figure 1.

The selection is done by choosing the corresponding check boxes placed near the design alternatives, respectively. Following, we describe the three steps in the AHP wizard that make up the adaptation of the AHP process.

Determining Relative Criteria After the user has selected the design alternatives that are to be compared using the AHP method, he/she is presented with a window displayed in Figure 2. This dialogue asks for the user's preferences on the QoS attributes. There are six pair-wise comparisons that need to be made.

[2]The Q-ImPrESS IDE, including the tool described in this paper, is available at `http://www.q-impress.eu/`

Assessment of Alternatives In the second step (as displayed in Figure 3), the AHP Wizard asks for the preferences of each comparable pair of data. It is expected that the user makes comparisons between individual values of a certain quality metric. For example, the user has to choose whether a response time of 8ms is better, and how much better than the one of 10ms. If the user needs more data on one specific value, such data is presented via a tool-tip that is displayed when hovering over that value.

Interpretation of Results Upon completion of the above steps, the results are displayed, as shown in Figure 4. The trade-off analysis results are shown as stacked bar graphs. Each bar is separated into portions that correspond to zones of the total score attributed to a particular QoS attribute, multiplied by the weight of the attribute, respectively. The total height of each bar corresponds to the score that has been computed for that design alternative. After completing all of the wizard's steps, the user can choose to return and make adjustments.

When the user is satisfied with the results, the wizard will export all of the entered data, and the analysis results, to an HTML file. This enables further processing of the results using any spreadsheet or word processing application. If the user wants to redo the analysis on the same alternative designs, the previously made choices, which have already been saved, are automatically loaded, such that they are available for further refinement. In this way, the user can adjust only the choices that have been changed, while preserving all the previously entered ones.

5. DESIGN DECISIONS

Our trade-off project's main requirement was to create a tool, within the Q-ImPrESS IDE, which should use the Analytic Hierarchy Process to help the system designer compare various design solutions, via the trade-off analysis between different quality prediction values. To meet these requirements, we have made several major design decisions. Firstly, we present an overview of these decisions by presenting the architecture of the tool. Secondly, we describe several of these decisions in more detail, as they were made to portray the design and development cycle of the tool. The constraints under which this project has been developed are: limited number of person hours and major changes of the underlying API during tool development.

5.1 AHP Wizard Architecture

In this subsection, we describe the architecture of the AHP Wizard, including the packages that have been used, and the link between the AHP Wizard and the rest of the Q-ImPrESS IDE. A component and connector view of the AHP Wizard is presented in Figure 5.

Not shown in this figure is the Eclipse Rich Client Platform, which is the underlying framework the the Q-ImPrESS IDE uses. Our tool is constructed using the SWT package of Eclipse RCP, to align the tool with the rest of the IDE tool and to take advantage of all the benefits of the Eclipse RCP.

The AHP Wizard has a fairly simple architecture of the graphical interface. Its graphical interface is based on the Eclipse JFace package (org.eclipse.jface.wizard) wizard implementation. This package contains the implementation of the WizardPage class, which we have extended with appropriate controls to provide the classes that describe each of the wizard dialogues that are used during the trade-off analysis procedure.

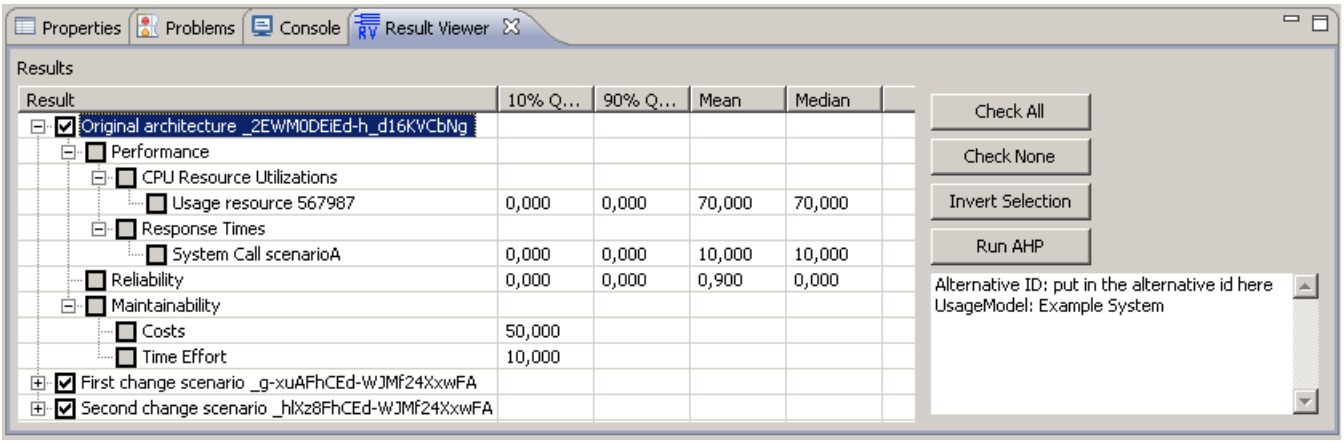

Figure 1: Result Viewer - a tool for presentation of quality prediction results.

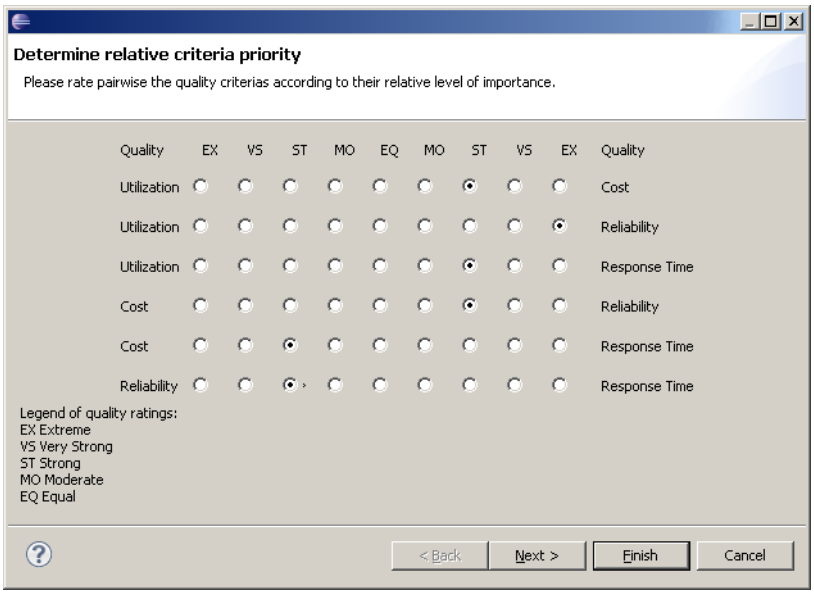

Figure 2: Pairwise rating of Quality-of-Service attributes

Besides the standard controls, we have also used the JFreeChart[3] package, to enable the stacked bar graph visualization of the results, and the Velocity Engine[4] to provide facilities for HTML output of the trade-off results.

The Q-ImPrESS IDE is connected to the AHP Wizard by invocation only; this is done through the Result Viewer, that is, the tool for viewing the QoS prediction results, which is presented in section 4.

On the other hand, the AHP Wizard is linked to the Q-ImPrESS IDE by using the facilities of the Q-ImPrESS IDE backbone. The backbone offers API calls that enable any part of the Q-ImPrESS IDE to read and write results in the, so called, Result Repository, where all of the quality prediction results reside in Result Models. Hence, an integration between the AHP tool and various Analysis Tools is established.

The AHP Wizard is also connected to the file system of the Eclipse Environment, to be able to ensure the data persistency over the currently selected choices. This persistency is done through a custom class developed for this purpose, and by serializing this class with a binary file. In essence, this class acts like a simple database.

5.2 Wizard Interface Paradigm

Considering the structure of the AHP method, one can easily notice that it consists of several steps. In addition, the method requires the user to input a large amount of data, which triggers the conclusion that a single window representation is inadequate for the job. Therefore, the wizard paradigm has proved itself to be the most convenient way to present multiple questions spread over multiple dialogues. We have also relied on a two-way flow of the wizard paradigm (that is, we rely on the wizard's possibility of returning to previous steps).

Our favorite paradigm is similarly used in many polling applications.

5.3 Fixed AHP Tree

The first step in applying the AHP method is to create a hierarchy tree of different parameters that are included in the trade-off process

[3] `http://www.jfree.org/jfreechart/`
[4] `http://velocity.apache.org/`

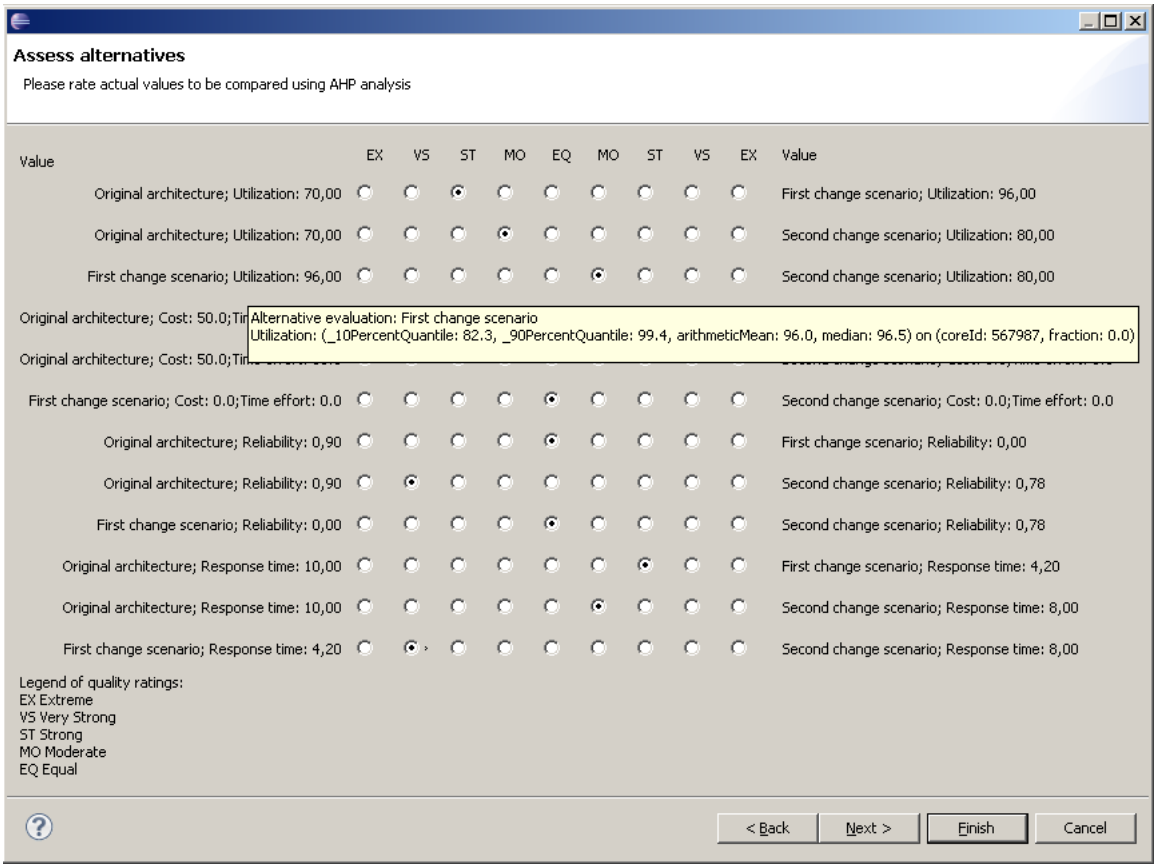

Figure 3: Pairwise rating of actual data.

(this is explained in more detail in [10]).

We have chosen to implement a fixed one-level hierarchy tree, which cannot be changed by the user. This decision has been considered appropriate as there is always a fixed set of inputs to the AHP method. The AHP Wizard reads out values for Response Time / Throughput, Reliability, Utilization and Cost / Effort and, for presentation purposes, calculates averages of inputs where multiple values are present. The individual values that produce the consolidated value can be accessed by hovering mouse over one of the consolidated values. It is up to the user to interpret individual values, compare them to individual values of another alternative and make a choice regarding which alternative is *better*.

We consider all QoS attributes independently of each other within flat one level tree hierarchy. This makes sense because all of the evaluations are made by the user so any implicit relationships within QoS attributes can be addressed during the rating phase of the AHP method.

This decision has drastically reduced the number of person hours required for completing the trade-off analysis project, entailing its feasibility.

5.4 Result Representation Using Stacked Bar Graph

After creating a prototype application, we noticed the need for the visual representation of the trade-off analysis results. The simplest and most straightforward way to represent the ratio between several values is either the pie chart or the bar graph. In our case, we have also wanted to represent the data that the values result from. The quality of a design alternative is calculated as a weighted sum over performance, reliability and maintainability values, hence we have chosen to represent the trade-off data via stacked bar graphs, which show almost all the data involved in the AHP method.

5.5 Storing the results in HTML format

HTML is an open standard that can be read on almost any platform. Motivated by this argument, we have searched and discovered that the Velocity Engine by Apache is an appropriate tool for generating the needed HTML, from a template. It also allows us to easily export the image of the generated graph to a portable and open format.

5.6 Persistence of the trade-off results

While developing the tool, we have noticed that there are many situations when the user has to go out of the tool, make a fix in the design alternative, rerun quality predictions, and then re-evaluate the trade-off between the new prediction values. To simplify this usage, we have implemented an entirely transparent system of saving the user's choices, which are automatically loaded when the user reopens the tool, assuming the same design alternatives are used. Unfortunately, there exists the pitfall of the tool not recognizing changes of prediction values within a design alternative. We will address this deficit in future extensions of the tool, as to alert the user on changes.

6. LESSONS LEARNED

Prior to developing the final version of the tool, we have devel-

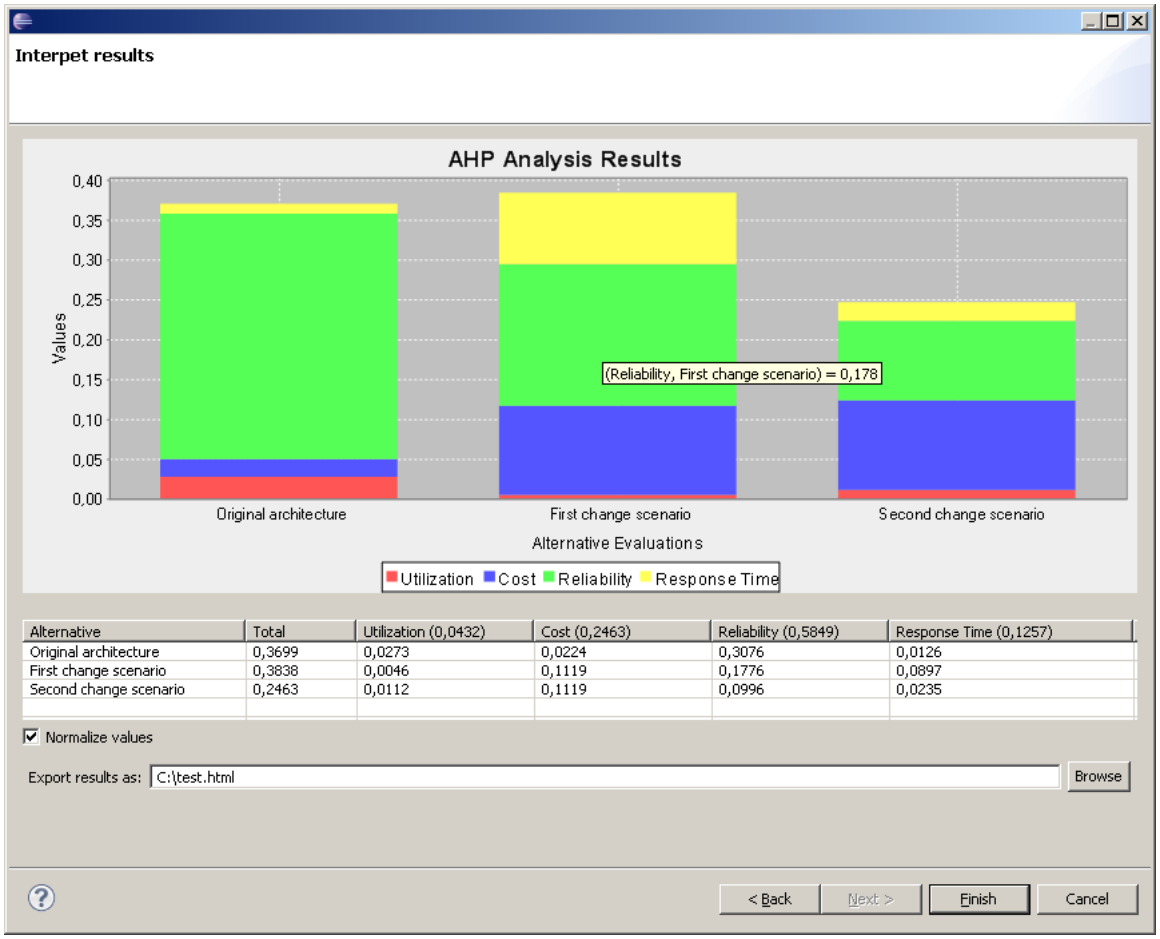

Figure 4: Displaying results.

oped a simple prototype with full AHP method functionality, yet weak on the usability side. However, this prototype has been very useful in determining many of the design decisions that we have just described, for the current, integrated version of the tool.

The tool development has been done mostly by iterative prototyping, which has met the time and person hour constraints of the project.

We have heavily relied on already existing packages, and thus were able to integrate mature and bug-proofed code, in our tool, which has also contributed to the timely completion of the tool.

7. RELATED WORK

With respect to trade-off analysis not many generic approaches exist, to our knowledge. Hence, we present only two other methods besides AHP: ATAM/CBAM and QFD. A well known approach in the architecture domain is the ATAM / CBAM approach [2]. In this approach, the ATAM uncovers the important design decisions, while the Cost Benefit Analysis Method (CBAM) attaches cost and benefits to these aforementioned decisions, thereby offering a method to rationalize which decisions (and therefore trade-offs) to make. A more generic approach to trade-off analysis like AHP is found in the Quality Function Deployment (QFD) approach [6]. The idea here is to satisfy requests and expectations from customers by translating them into design targets and major quality assurance points. By offering a comprehensive traceability matrix, individual decisions can be traced back to the customer priorities that drive them. Hence, trade-offs are also made very explicit in this approach.

There exist many different architectural analysis approaches for all kinds of system qualities. Each quality has more or less its own research community, making integration of different approaches for different qualities far from trivial, as is the case for the Q-ImPrESS IDE. There exist two flavors of architectural analysis: quantitative analysis and qualitative analysis. In quantitative analysis, as used in the Q-impress IDE (e.g. PCM [4], Klaper [5]), the result of the analysis is a quantified metric, which allows for estimating how *much* better a design alternative is over another one, with respect to that metric. With qualitative analysis, such comparisons are not possible, as it is only known that one alternative is better than another, but the extend is unknown. However, typically the qualitative approaches take considerable less effort. Hence, for such an analysis, one has to make a trade-off between precision and effort.

As mentioned before, our use of AHP for trade-off decision making in software architecture is not new. Zhu et al. [10] employed AHP to determine the sensitivity of architectural alternatives to different criteria. Svahnberg and Wohlin [9] use the method to build consensus among different stakeholders with respect to different architectural alternatives. Al-Naeem et al. [1] demonstrate how the method can be used to optimize architectural decision making when having multiple architectural alternatives. Compared to these three different studies, our approach is new in the sense that it directly in-

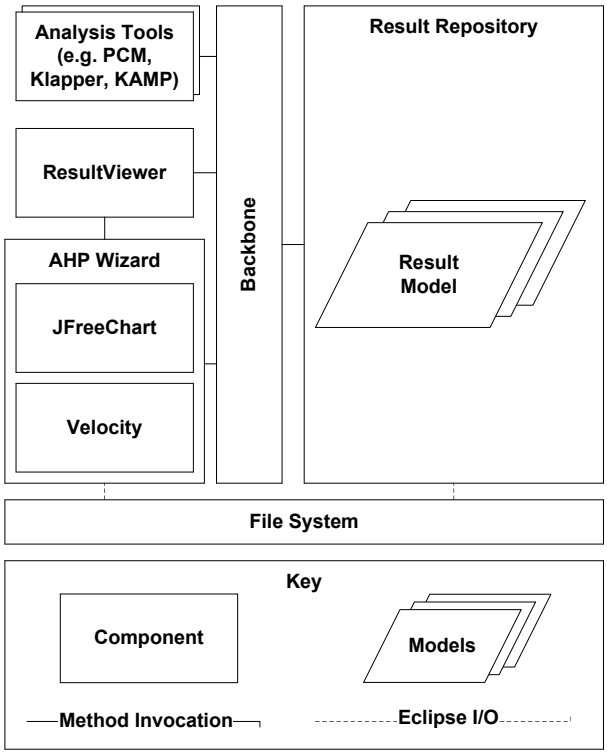

Figure 5: Component & connector view of the AHP Wizard.

tegrates with specific analysis tools, as provided in the Q-ImPrESS IDE, which is something not done previously, as the metrics used before to asses an architectural alternative were estimates provided by experts and did not come from specific analysis tools.

8. CONCLUSION

In this paper, we have presented a tool that provides support for the AHP-based trade-off analysis of quality prediction values for multiple design alternatives of SOS. We presented how the generic AHP method was adapted to ease its use in the Q-ImPrESS IDE. Furthermore, we presented the main issues faced during the tool implementation, and their remedies. In particular, we presented a comprehensive list of major decisions taken during the development process, together with the lessons learned from this project. Future work on the tool includes notifying the user when no longer the previous used comparisons can be reused, as either the quality criteria changed, or the set of alternatives considered changes. Another direction for future work we would like to investigate is how the AHP method for trade-off analysis compares to other trade-off methods.

9. REFERENCES

[1] T. Al-Naeem, I. Gorton, M. Babar, F. Rabhi, and B. Benatallah. A quality-driven systematic approach for architecting distributed software applications. In *Software Engineering, 2005. ICSE 2005. Proceedings. 27th International Conference on*, pages 244 – 253, May 2005.

[2] L. Bass, P. Clements, and R. Kazman. *Software architecture in practice 2nd ed.* Addison Wesley, 2003.

[3] S. Becker, M. Hauck, M. Trifu, and K. Krogmann. Reverse engineering component models for quality predictions. *Proceedings of the 14th European Conference on Software Maintenance and Reengineering*, March 2010.

[4] S. Becker, H. Koziolek, and R. Reussner. Model-based performance prediction with the palladio component model. In *WOSP '07: Proceedings of the 6th international workshop on Software and performance*, pages 54–65, New York, NY, USA, 2007. ACM.

[5] V. Grassi, R. Mirandola, and A. Sabetta. From design to analysis models: a kernel language for performance and reliability analysis of component-based systems. In *WOSP '05: Proceedings of the 5th international workshop on Software and performance*, pages 25–36, New York, NY, USA, 2005. ACM.

[6] S. Haag, M. K. Raja, and L. L. Schkade. Quality function deployment usage in software development. *Commun. ACM*, 39(1):41–49, 1996.

[7] T. L. Saaty. *The Analytic Hierarchy Process: Planning, Priority Setting, Resource Allocation*. McGraw-Hill International Book Co., 1980.

[8] J. Stammel and R. Reussner. Kamp: Karlsruhe architectural maintainability prediction. In *Proceedings of the 1. Workshop des GI-Arbeitskreises Langlebige Softwaresysteme (L2S2): "Design for Future - Langlebige Softwaresysteme"*, pages 87–98, 2009.

[9] M. Svahnberg and C. Wohlin. Consensus building when comparing software architectures. In *Proceedings of the 4th International Conference on Product Focused Software Process Improvement (PROFES 2002)*, volume 2559, pages 436–452. Springer, December 2002.

[10] L. Zhu, A. Aurum, I. Gorton, and D. R. Jeffery. Tradeoff and sensitivity analysis in software architecture evaluation using analytic hierarchy process. *Software Quality Journal*, 13(4):357–375, 2005.