Electronic version of an article published in **STRAW '03. Second International SofTware Requirements to Architecture Workshop, Portland, Oregon, 2003; pp.142-149**

http://www.empress-itea.org/

An Experience-Based Approach for Integrating Architecture and Requirements Engineering [#]

B. Paech[*], A.Von Knethen[*], J.Doerr[*], J.Bayer[*], D. Kerkow[*], R. Kolb[*], A. Trendowicz[*], T. Punter[*], A. Dutoit[+]
[*]Fraunhofer IESE, Sauerwiesen 6, 67661 Kaiserslautern, Germany
[+]TU München, Institut für Informatik,
{paech, vknethen, doerrj, bayer, kerkow, kolb, trend, punter}@iese.fraunhofer.de, dutoit@in.tum.de

Abstract

Deriving requirements and architecture in concert implies the joint elicitation and specification of the problem and the structure of the solution. In this paper we argue that such an integrated process should be fundamentally based on experience. We sketch an approach developed in the context of the EMPRESS project that shows how different kinds of experience-based artifacts, such as questionnaires, checklists, architectural patterns, and rationale, can beneficially be applied.

1. Introduction

The last few years have seen a growing awareness of the requirements engineering community for architectural issues and vice versa. Several authors have argued convincingly for the tight interdependencies between functional requirements (*FRs*), non-functional requirements (*NFRs*) and architectural options (*AOs*) that need to be made explicit early, e.g., [1], [2].

The design of an architecture aims at creating a software solution for the problem given in the requirements specification. In the requirements specification, the problem is elicited and documented using concepts from the problem domain. An architecture sketches the solution at a high level of abstraction. This means that the problem must be expressed in terms of concepts from the solution domain (i.e., the programming domain). This is a creative activity that is not well supported by current software development approaches.

In this paper, we propose an approach that supports the elicitation, specification and design activity by providing *experience* in terms of questionnaires, checklists, architectural patterns and rationale that have been collected in earlier successful projects and that are presented to developers to support them in their task.

[#]The research for this paper has been partly funded by the EUREKA-ITEA projects "EMPRESS" (ITEA 01003) and "CAFÉ" (ITEA 00004)

The approach uses a refinement graph, checklists and questionnaires to capture important NFRs more precisely. In addition, it uses architectural patterns for reusing AOs and for evaluating them against a specific set of requirements. Furthermore, it uses traceability and rationale management to make explicit the decision making involved in a joint specification and design of FRs, NFRs and AOs.

The paper is structured as follows: First, we sketch the fundamental issues to be solved in integrating RE and architecture development, and how these are covered by related work. Second, we discuss the foundation of our approach in terms of a metamodel that describes the basic concepts we are dealing with, such as quality attributes, metrics, and NFRs. Third, the integrated process with input and output documents is described. We conclude with a discussion of how well our approach deals with the fundamental issues identified.

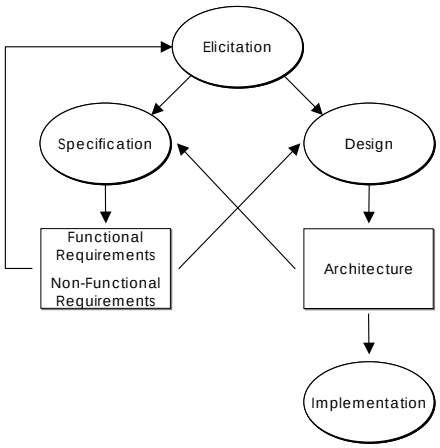

Figure 1: General process of integrating architecture and requirements

2. Fundamental Issues

Figure 1 shows the general process of integrating the architectural decision process into the requirements engineering process.

It comprises the relevant activities of the software engineering process, namely an iteration of requirements elicitation, specification, and design that produces FRs, NFRs and AOs. These are subsequently implemented.

As exemplified by the approaches presented at STRAW 2001, there are many different ways to support these activities. They mainly solve the following fundamental issues:

- **Issue 1 – Views of different stakeholders in the elicitation of NFRs, FRs, and AOs**: How to identify the essential NFRs, FRs, and AOs and different views of different stakeholders? How to negotiate conflicts? What is a sufficient level of abstraction for these discussions? One possible support for negotiation is given by the WinWin approach [2][3][4].
- **Issue 2 – Identification of dependencies among FRs, NFRs, and AOs**: How to describe NFRs, FRs, and AOs such that dependencies can easily be identified? In several approaches, goal graphs are used for specifying NFRs and FRs and their dependencies. There is much less agreement on describing AOs, e.g., Use Case Maps [5], agent-oriented goal graphs [6], the CBSP approach [4], or social organizations [7].
- **Issue 3 – Assessment of how well different AOs address a specific set of FRs and NFRs**: How to capture and support the decision making involved in specifying FRs, NFRs and AOs? Typically, concepts from *rationale management* [8] are used to make explicit questions to be solved, options for their solutions, criteria to evaluate the options and assessments of the options against these criteria. For example, goal graphs are used to capture criteria (business goals) and issues (NFRs and FRs), AOs and their assessments [5]. Another example is the Concordance Matrix to capture assessments of the architectural relevance of FRs and NFRs [4]. Also, SEIs Architecture Tradeoff Analysis Method (ATAM) captures criteria (quality attributes, business goals), issues (risks), options (architectural views), and assessments (utility tree). The Cost Benefit Analysis Method (CBAM) is used to refine the ATAM results with cost, benefit (criteria, options) [3].

As argued in the introduction, however, the design of an architecture is a creative task. It involves much judgment and heuristics on the importance of NFRs and FRs and different AOs. Thus, it is error-prone (e.g., guesses about how well an architecture meets a set of NFRs can be wrong) or expensive (e.g., when using a prototype realization of the architecture to experimentally assess the suitability of the architecture). Moreover, it can only be learned through experience and apprenticeship. Hence, leveraging off past experience can help these challenges to be addressed. This raises another issue:

- **Issue 4 – Representation of past experience to facilitate issues 1-3**: How can one capture and use experience on FR, NFRs, AOs, their dependencies and their assessments? Such representations must not only include the AOs under consideration, but also sufficient knowledge for their selection and application. This includes the context in which they can be used and the trade-offs they entail. Architectural styles, for example, are used to capture typical AOs, a correlation catalogue to capture typical assessments [7].

The last issue is rather implicitly treated in many approaches. In contrast, we have put most emphasis on identifying how experience can support the integrated process.

3. Our Approach

In the following, we present our approach for capturing experience to support the integrated elicitation, specification of NFRs and FRs and design of architecture. First, we explain the fundamental concepts in terms of a metamodel. Second, we sketch the process and the products. We illustrate the process and the products with a case study dealing with a mobile, interactive application to allow users monitor production activities, manage physical resources and access information. This case study is based on a real system and was provided by Siemens in the context of the Empress project.

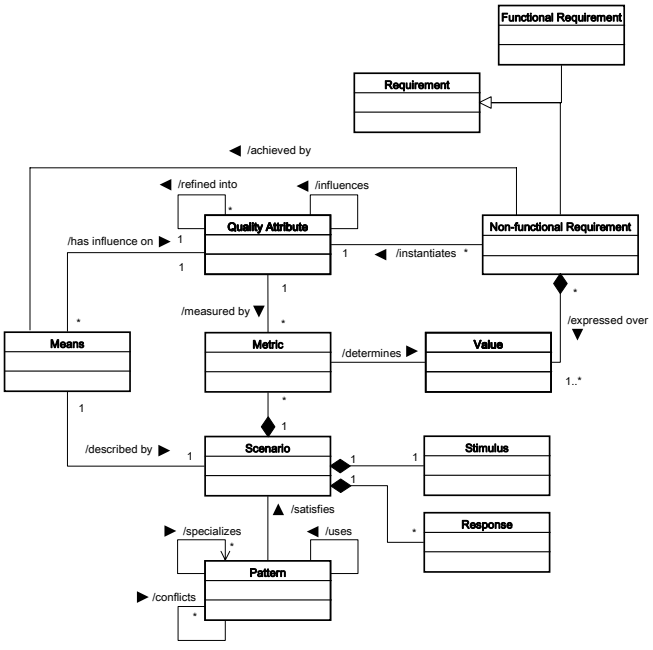

Figure 2: The metamodel

3.1 Foundation

Our integrating approach is based on a metamodel that describes the main concepts we are dealing with (see Figure 2).

- *quality attribute* (*QA*) is a non-functional characteristic of a software product or process. We distinguish between high-level QAs (i.e., efficiency, maintainability, reliability, usability, and portability) and refining QAs of these attributes. The high-level QA "efficiency" can, for example, be refined into "time behavior" and "resource utilization", "time behavior" can be refined into "workload" and "response time". In addition, QAs can have positive or negative *influences* on each other, e.g., if the "workload" is higher, the "response time" will increase (negative influence).
- To make explicit the distinction between knowledge about QAs gained in experience and the quality to be achieved in a specific project, we use the term *NFR* to describe the latter. A NFR is an *instantiation* of a QA that is created by determining a *value (range)* for a *metric* associated with the QA. For example, the NFR "The database of our new system shall handle 1000 queries per second." instantiates the QA "workload of database". The value is determined based on an associated metric "Number of jobs per time unit".

The distinctive feature of this metamodel is that we distinguish problem-oriented refinement from solution-oriented refinement of QAs. The latter is made explicit in terms of means which mediate between QAs and patterns.

- *Means* are principles, techniques, or mechanisms that facilitate the achievement of certain qualities in a software architecture. They are abstract patterns that capture a way to achieve a certain quality requirement, but are not concrete enough to be used directly (i.e., they have to be instantiated as patterns). Means are described by *scenarios*, which consist of stimulus and response, and a metric. For example, a scenario for the NFR mentioned above is "object creation throughput must be fast", where the stimulus is "object creation", the response is "throughput" and the metric is "number of objects created per second".
- A *pattern* is used to document Aos. Pattern help designers in creating architectures by providing solutions for recurring problems in the design of software architectures. The pre-defined solutions have proven to be beneficial in certain situations. As they have been applied repeatedly, their impact on a software architecture is known. Patterns are chosen to satisfy the scenarios. They can be refined through *specializations*. For example, the pattern "layered architecture" can be specialized into "strictly layered architecture" and "loosely layered architecture". Furthermore, if a pattern *uses* another pattern, the used pattern is applied to create the using pattern. With this mechanism, collaborating patterns can be used to form higher-level patterns. Two patterns can also be in *conflict*, e.g., the "client server" and "layered architecture" patterns cannot be applied at the same time.

The following sections describe how these concepts are used within our approach.

3.2 Experience-Based Process

Figure 3 gives an overview on our experience-based process of integrating architectural decision making into the requirements engineering process.

In the following, the different activities of our process are listed. The overall process is iterative, that means within each activity and between the activities iterations are probable and necessary. Products consumed and produced by the activities of the process are explained in more detail and illustrated with examples in the following. sections.

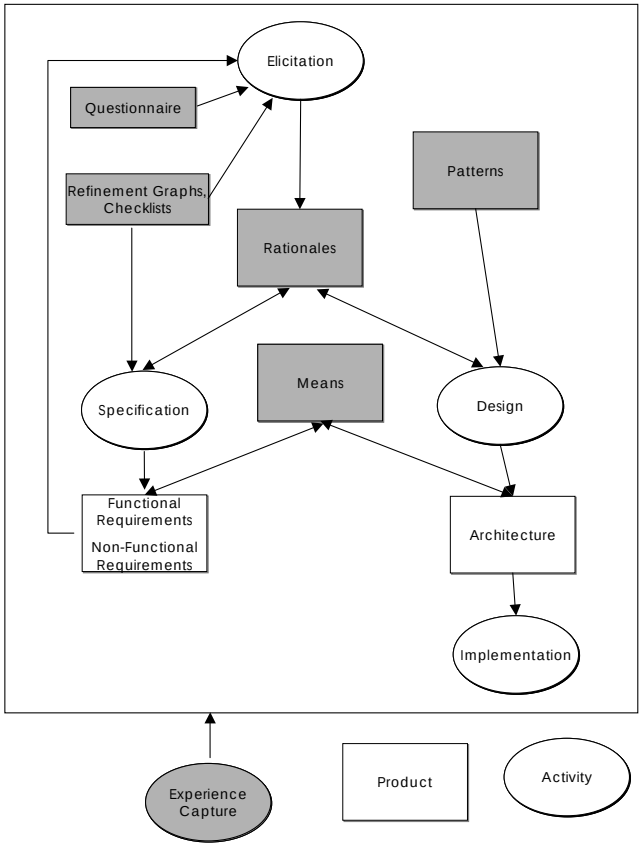

Figure 3: The experience-based process

- **Elicitation:** During the elicitation, the customer has to prioritize the QAs at the highest level of abstraction for the system to be developed. A questionnaire is used for this purpose. Then, QAs with the highest priorities are refined with the help of checklists. Refinement graphs for the high level QAs are the foundation of all checklists. We distinguish different types of checklists. Each checklist focuses on a certain refinement aspect (e.g., problem-refinement, solution-refinement, dependencies between QAs). The rationale for specific estimates for the NFR (e.g. maximal load) is captured.
- **Specification:** During the specification, measurable NFRs will be documented in a requirements document. Checklists guide this activity. We use a requirements template that allows different NFRs to be described at different places in the document. NFRs, for example, that are expressed over FRs are explicitly stated together with the FR. We use Use Cases and Use Case descriptions to describe FRs (our approach for describing FRs for embedded systems has been developed in the QUASAR project [9]). NFRs (e.g., response time requirements) are explicitly stated in the Use Case descriptions.
  Furthermore, concrete means to achieve the NFRs are identified by using the assessments of their suitability documented in a refinement graph. The rationale for a chosen means is captured.
- **Design:** During the design, requirements that have an effect on the architecture are selected. In addition, the principal structure of the system is refined based on the requirements and the means and pattern catalogue. In the following, the existing architecture is iteratively refined based on requirements and the catalogue. After each refinement step, the architecture is assessed concerning their non-functional properties. The rationale for chosen means and patterns is captured.
- **Experience Capture:** During the performance of a project, experiences are collected and consolidated to improve the questionnaire, refinement graphs and checklists and the patterns and means catalogue.

3.3 Questionnaire for Prioritization

For the prioritization of QAs at the highest level of abstraction, a standardized questionnaire is used. The questionnaire elicits wishes and facts concerning the development context of the customer and relates them to a selection of the QAs defined by ISO9126 [10]: we selected maintainability, efficiency, usability, and reliability in our case study.

In the following, we describe at first how the questionnaire was developed and then how it can be applied.

To develop the scales of the questionnaire, in a first step, potential scale items were generated. For this purpose, we phrased a set of 120 statements containing wishes and facts, which a person involved in a system development project would express. The statements covered the complete set of second level QAs (ISO 9126) of the high level QAs mentioned above.

Once the statements were generated, they were presented to eight software quality experts. These experts judged, whether a customer that needs a certain QA would agree to each statement. A 1-5 rating scale was used for the judgment. The experts were – as usual in scale development [11] – asked not to rate their own project context, but rather to judge based on their personal experience, how favorable each item is with respect to the QA of interest.

In a next step, items with the highest mean and lowest variance (high interrater reliability) were selected and assembled to a 30 items questionnaire. As response scale, a 1-5 Likert scale was chosen, of which 17 statements covered facts of the current project (strongly agree – strongly disagree), and 13 statements covered wishes for future conditions (very important – very unimportant).

The determination of the mean value of the statements affecting one QA enables to build a rank order of these. The one with the highest ranking is the most important attribute for the current system development project and should receive the greatest deal of attention. The prioritization is of special interest in case of limited requirements engineering resources and allows focusing the requirements engineers' energies on the most important high-level QAs. This priorization questionnaire was applied in the case study. It did not confirm the prior expressed expectations of the customers. A closer analysis showed, that the customers tended to rate such quality aspects as most important, that were difficult for them to handle, namely "efficiency", instead of naming the most important aspects for the success of the project in scope. The results of the prioritization questionnaire ranked "maintainability" as most important. The customer confirmed the correctness of this result.

3.4 Refinement Graph

A refinement graph (also called quality model) instantiates parts of our metamodel. It describes typical refinements of high-level QAs into more detailed QAs, metrics, and means. In addition, it describes relationships between different QAs. Therefore, it captures experience of previous projects. Our refinement graph is similar to the goal graphs of e.g. [6], but emphasizes dependencies. Figure 4 gives an example for such a refinement graph for the QA "efficiency". White rectangles represent QAs at

different levels of detail. Ovals represent metrics that measure certain QAs. Grey rectangles represent means to achieve certain QAs.

Four types of relationships can be found in such a refinement graph. The metamodel in Figure 2 describes the general types of relationships.

- A QA, such as "efficiency" is *refined into* more detailed QAs, such as "time behaviour" and "resource utilization".
- A means *has influence on* a QA, i.e., it is used to achieve the QA, e.g., "load balancing" is used to achieve "workload distribution".
- A QA is *measured by* a metric. For example the "workload" can be measured by the metric "number of jobs per time unit".
- A QA can be positively or negatively *influenced by* another QA. If the "workload", for example, is higher, the "response time " will increase (negative influence).

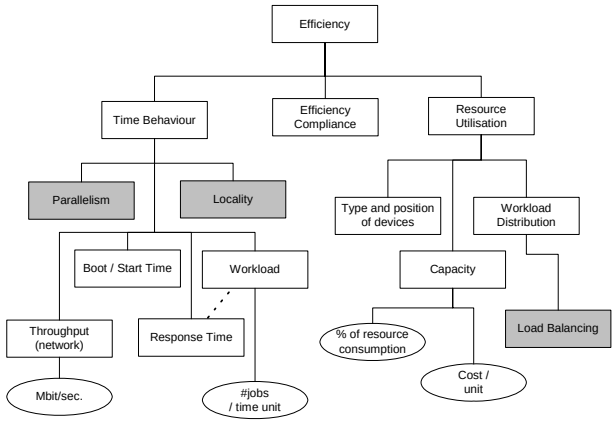

Figure 4: Refinement graph for efficiency

Our approach provides a default refinement graph that can be used without adaptations by a company. Reasons for this can be a lack of time or money. We recommend tailoring the refinement graph to the context of each company and project. Alternatively, a company might have an own refinement graph that shall be used. In this case, it is very important to agree on the meaning of the different QAs in this graph. Our recommendation is to build a refinement graph together with the company in a workshop. By doing so, the refinement graph benefits from the already integrated experience of our default refinement graph and it is tailored to the project and company. So far, we defined default refinement graphs for the QAs "efficiency", "reliability", and "maintainability". NFRs are elicited for each QA and relationships between NFRs and FRs are established via the checklists.

A mechanism to capture the experience of multiple projects and store the various refinement graphs is also developed as part of the ITEA EMPRESS project. This so-called *Prometheus* approach (Probabilistic Method for early evaluation of NFRs) is described in [12].

3.5 Checklists

Based on the information included in the refinement graph, we developed checklists that focus on different aspects of a high-level QA. We distinguish for each high-level QA between: (1) initialization checklists, (2) refinement checklists, and (3) dependency checklists. All checklists are described in more detail in the following. Again, in the other approaches for integrating RE and architecture we have not found something similar to checklists. They help to make the experience captured in the refinement graph directly applicable in workshops.

Initialization checklists are defined that capture everything that has to be decided before NFRs are refined. There are two types of initialization checklists that are used in our process: a general initialization checklist and specific high-level QA checklists.

The general initialization checklist includes aspects of the following categories:

- Organizational aspects (e.g., domain knowledge required)
- Technical issues (e.g., notations required)

Figure 5 depicts an extract of such a general initialization checklist.

1. **Supplier** *(project issues->process stakeholders)*
- Do you expect a certain kind of organization? (e.g., number of employees, size of company, lifetime of company)
- Do you expect a certain domain experience? If yes, specify the level of experience! (e.g., 2 years experience / number of developed systems in the domain)

2. **Process and documents**
- Do you have certain constraints towards the system development process?*)*
  a. Conformance to standards (e.g., ISO 9001, IEEE …) *(project issues->process requirements || documentation requirements)*
  b. Special activities (e.g., audits/reviews) *(project issues->process requirements*
  c. Documentation (e.g., specification documents) *(project issues->documentation requirements)*
  d. Notation (e.g., statecharts) *(project issues-> documentation requirements)*

Figure 5: Extract of general initialization checklist

Initialization checklists include a set of questions. To support answering the questions, examples are given in brackets. Italic formatted comments describe at which place in the requirements document, the information should be stated. Examples for NFRs concering organizational experience are:

- "At least 3 years of experience in maintenance is required (the longer the better)."

- "Project experience with wireless networks is required."

An excerpt of a specific initialization checklist for the high level QA "efficiency" is given in Figure 6. This structure of the checklist corresponds to the structure of the other initialization checklist. In our case study, there were no specific NFRs concerning the organizational experience regarding efficiency.

1. **Supplier** (*project issues->process stakeholders)*
- Do you expect a certain experience in building time critical systems? If yes, specify the level of experience! (e.g., had a similar project before, has special qualification (training, certification)).
- Do you expect a certain experience in building systems with resource limitations? If yes, specify the level of experience! (e.g., had a similar project before, has special qualification (training, certification))
2. **Process and documents** (*project issues->process requirements || documentation requirements)*
- Are there any laws or standards regarding efficiency your system will have to adhere to? If yes, specify the requirement!

Figure 6: Excerpt of efficiency initialization checklist

l NFRs, refinement checklists are used to elicit specific measurable NFRs. Refinement checklists are specific for high-level QAs (e.g., efficiency). In case of efficiency and reliability requirements, we recommend creating Use Cases to identify concrete NFRs. An excerpt for the refinement checklist for throughput NFRs is given in Figure 7. Again, text in italics indicates the place to document the NFR in a given document structure.

6. **Elicitation of Throughput NFRs** *(FRs-> UC description -> NFRs -> throughput // NFRs->efficiency ->throughput)*

For each network element in the system architecture:

- Go through each Use Case: Identify the UC-steps and exceptions involving data transportation on this component. Think of an **average** scenario of the Use Case
  - How much data has to be transported by this component?

  Specify the throughput NFRs on the network element.
- Go through each Use Case: Identify the UC-steps and exceptions involving data transportation on this component. Think of a **maximum** usage of the Use Case
  - How much data has to be transported by this component?

Specify the throughput NFRs on the network element.

Figure 7: Excerpt of refinement checklist for throughput

Measurable efficiency NFRs that were elicited by using the refinement checklists in the case study are for example:

- In a maximum usage, 8 people must be able to download a document (about 1 MB) within 10 sec. via the WLAN (6.4 Mbit/s).
- The PDA must be able to handle 60 alarms (coming from machines) at the same time.
- The memory of the database server must at least have a capacity of 512 MB.

While eliciting the NFRs, dependencies to other NFRs and architectural decisions are checked by using a dependency checklist. Figure 8 depicts an excerpt of the efficiency dependency checklist.

After applying these checklists, conflicts between NFRs and solution alternatives are documented. If concrete solutions were specified, also the rationale for the decision is documented. In our case study, a conflict appeared between the following two NFRs:

- "In a maximum usage, 8 people must be able to download a document (about 1 MB) within 10 sec. via the WLAN (6.4 Mbit/s)."
- "The WLAN supports 10 Mbit/sec."

In this case, the net throughput of the WLAN might be not sufficient for the first requirement. This conflict was documented.

The NFRs have been expressed separately. Now check for the NFRs affecting the same architecture component and instantiating dependent quality attributes:

- Consider first dependencies between refinements of efficiency: Is it possible to fulfill all NFR at the same time (e.g. response time and workload)? If not, high-light the conflict.
- Consider then all dependencies: Is it possible to fulfill all NFR at the same time (e.g. response time and maintainability)? If not, high-light the conflict.

Figure 8: Excerpt from efficiency dependency checklist

3.6 Means and Architectural Patterns

The general dependencies between means and patterns are captured in a separate catalogue. This catalogue is used as follows. A designer working on a certain component or (sub-) system chooses the architectural relevant FRs, as well as the NFRs. The NFRs are then used to select appropriate means. This is done by comparing the scenarios associated with the means with the requirements. Once the means are selected, the patterns that specialize the respective means are selected from the catalogue. This is again is done by comparing the scenarios related to the patterns with the requirements. The selected patterns are instantiated to support the design.

3.7 Rationale

The refinement graph and the catalogue capture general relationships between QAs, means and patterns. The choice of a specific pattern requires detailed evaluation of the means and the patterns against the relevant requirements. We capture this evaluation in terms of rationale that can then be used to refine the refinement graphs and the catalogue.

The designer documents the selection of means with an assessment matrix for each subsystem under consideration (see Table 1). The rows of the matrix

represent the selected means. The columns of the matrix represent the requirements that are relevant to the subsystem under consideration. Each cell denotes whether a specific means makes it easier or more difficult to realize the corresponding requirement with the symbols "+" and "-" and a reference to the scenario that was used to generate the value. If the means has no impact on the requirement, the cell is left empty. Once the matrix has been filled out, the designer identifies potential conflicts between selected means. While the designer can select alternate means in order to reduce the number of conflicts, in general, however, the potential conflicts cannot be completely eliminated. The remaining conflicts are documented by annotating the cells (i.e., means x requirement x scenario) that are involved in the conflict for further consideration during the next step.

	FR1	FRn	Efficiency	Maintainability
Locality			-	+
Load balancing			+	-
Caching			+	-
Concurreny			+	-
Sharing			+	-

Table 1. High-level assessment matrix for detecting conflicts among means

The patterns are selected by comparing the scenarios related to the patterns with the requirements. For each means, the designer builds a new assessment matrix. The rows represent the candidate patterns selected with the scenarios. The columns include the requirements addressed by the means. When the means under consideration is involved in a conflict, the columns in the higher-level matrix that are negatively affected by the means are reported into the lower-level matrices. The designer uses the scenarios that result in negative assessments in the higher-level matrix to select a set of architectural patterns, hence addressing the relevant requirements and resolving the potential conflict.

This two-level approach for documenting trade-offs between options is similar to the rationale capture of designing services from user tasks described in [13]. The use of an assessment matrix enables the designer to summarize the rationale behind the selection of means and patterns and their evaluation with scenarios. Using a two level selection process reduces the size of the matrices that the designer has to work with and the total number of cells that need to be considered. By identifying conflicts in the higher-level matrix and reporting conflicting columns in the lower-level matrices, the designers focuses only on the relevant interactions between means and attempts to address those during the pattern selection and instantiation. Thus, the distinctive feature of our rationale capture is the detailed guidance we give for decision making.

4. Conclusion

We have presented a comprehensive approach covering the issues identified in section 2.

- Issue 1: The different views of the stakeholders are elicited and negotiated through the prioritization questionnaire, different view-oriented checklists and the rationale-based discussion. The distinction between QAs and means helps to keep the discussion on an adequate level of abstraction. This is achieved by separating problem refinement from solution refinement.
- Issue 2: Typical dependencies between QAs are captured in the refinement graphs. Concrete dependencies are elicited with the help of checklists and are captured in the rationale matrices. We use patterns to document AOs and Use Cases to document FRs. We use a requirements template that allows different NFRs to be described at different places in the document. NFRs, for example, that are expressed over FRs are explicitly stated together with the FRs. However, we have not yet worked on an intuitive representation of the dependencies between patterns and Use Cases.
- Issue 3: The relationships between AOs, FRs and NFRs are covered by our rationale matrices.
- Issue 4: As described in detail, we capture and use experience in terms of the questionnaire, the refinement graphs, the checklists, the patterns, and the rationale.

Of course, there are still many issues to be solved, in particular a full-scale case study. So far, we have used this approach together with our cooperation partners from Siemens to elicit and specify the FRs, NFRs, means and metrics. In a 2 day workshop its was possible to define a measurable and a more complete set of NFRs in comparison to ad-hoc approaches. In addition, the relationships between FRs and NFRs were clear. The choice of the patterns will be performed in the near future.

Till the end of the year, we plan to address the following questions:

- Package experience for different QAs from literature, in particular the catalogues for means and patterns.
- Find suitable architecture descriptions that facilitate the assessment of the dependencies between requirements and AOs.

So far, we have not investigated the utilization of problem frames (as a further instance of documented experiences). That would correspond to capturing typical FRs in the refinement graph. This would generalize our work from the domain of embedded systems – which is the focus of EMPRESS – to other domains.

Acknowledgements

We acknowledge the ITEA project EMPRESS for partly funding our research. Furthermore, we want to thank all partners in the ITEA project EMPRESS that contributed to our research. In particular, we want to thank Ricardo Jimenez Serrano (Siemens) for providing a case study to validate our approach.

References

[1] B. Paech, A. Dutoit, D. Kerkow, A. von Knethen: „Functional requirements, non-functional requirements and architecture specification cannot be separated – A position paper", REFSQ 2002

[2] In, H., Boehm, B.W., Rodgers, T., Deutsch, W., "Applying WinWin to Quality Requirements: A Case Study", ICSE 2001, pp. 555-564, 2001

[3] In, H., Kazman, R., Olson, D., "From requirements negotiation to software architectural decisions", STRAW 2001

[4] Egyed, A., Grünbacher, P., Medvidovic, N., "Refinement and evolution issues in bridging requirements and architecture – the CBSP approach", STRAW 2001

[5] Liu, L., Yu, E., "From requirements to architectural design – using goals and scenarios", STRAW 2001

[6] Gross, F., Yu, E., "Evolving system architecture to meet changing business goals: an agent and goal-oriented approach", STRAW 2001

[7] Kolp, M., Castro, J., Mylopoulos, J., "A social organization perspective to software architectures", STRAW 2001

[8] Dutoit, A., Paech, B., "Rationale management", in Handbook of Software and Knowledge Engineering, World Scientific Publishing, 2002

[9] von Knethen, A., Paech, B., Houdek, F., Kiediasch, F., "Systematic Requirements Recycling through Abstraction and Traceability", Int. Conf. RE 2002

[10] ISO/IEC 9126-1:2001(E), "Software Engineering - Product Quality - Part 1: Quality Model", 2001

[11] Trochim, W. M. K., "The Research Methods Knowledge Base", Atomic Dog Pub Inc., Cincinnati, 2001

[12] Punter, T., Trendowicz, A., Kaiser, P., "Evaluating evolutionary software systems", PROFES 2002

[13] Dutoit, A., Paech, B., "Rationale-based Use Case specification", Requirements Engineering Journal, special issue REFSQ 2001, 2002

[14] E. Dincel, N. Medvidovic, A. van der Hoek, "Measuring Product Line Architectures", in: Int. Workshop on Product Family Engineering, Bilbao (S), 2001