Problem-Solution Mapping for Evolution Support of Software Architectural Design

Matthias Riebisch

Ilmenau University of Technology
matthias.riebisch@tu-ilmenau

Abstract: Software architectures constitute a crucial factor for the development and evolution of software systems because they have to sustain their quality properties like flexibility, scalability, or security. A software architecture has to reflect the transition from the problem space with quality goals and requirements on one side to the solution space with technical solutions on the other side. A mapping between goals and solutions shall enable long-term evolution of the systems by expressing design knowledge and fundamental decisions explicitly. The gap between the fields of requirements engineering, software architectural design and software quality has not yet been closed. This paper discusses the Goal Solution Scheme, which maps quality goals and goal refinements to architectural principles and solutions. Its application in forward and re-engineering activities is shown, as well as its establishment. The concept has been evaluated in various projects including industrial case studies.

1 Introduction

Enterprise software systems have to fulfil highly complex requirements. They have to support business processes with a long lifetime, while they frequent changes have to be performed – for example regarding business rules, organizational optimization or technical platform. There is a strong need for long-term evolution of these software systems, because an end of support or a replacement by a newly developed system would cause extreme risks. Even for embedded software systems because of their increasing complexity there is a growing need for evolution. For both classes of software systems, quality requirements such as flexibility, scalability, usability, and security bear even more risk than functional requirements, because they can hardly be implemented after making the major design decisions.

Software architectures play an important role for complex software systems. They help to manage the systems' complexity, for example by organizational support for the development process, and by a representation of the design knowledge as well of the most crucial decisions. Software architectures reduce the developments risks by enabling early assessments of the fundamental decisions on technical solutions, especially for those regarding the quality properties of a software system. They represent the transition from goals and requirements in the so-called problem space to technical implementations in the so-called solution space.

Regarding evolution, the role of software architecture is two-fold. A software architecture enables evolution because it safeguards basic decisions as well as it supports changes in a well-organized way. On the other hand, a software architecture has to be maintained during the sequence of changes to prevent the so-called architectural decay.

In this paper, an explicit mapping from goals and requirements to architectural solutions is established by dependencies. An explicit representation of these relationships enables comprehension, evaluation, and utilization by tools, similarly to other model-based approaches. For this mapping, the Goal Solution Scheme has been introduced. In the paper the general structure and the utilization of the Goal Solution Scheme is discussed, together with its establishment and evaluation.

2 Related Works

Several architectural methods emphasize the analysis of **non-functional requirements** [HN00] and the design considering them, for example Bosch's Quality Attribute-based Software Architectural (QASAR) method [Bo00] and the Attribute-Driven Design (ADD) method [BK02]. **Goal-oriented requirements engineering** approaches–such as NFR and i*–model so-called softgoals and their refinement in a Softgoal Interdependency Graph (SIG). The SIG has been standardized in the User Requirements Notation (URN) [CP09]. These approaches facilitate a goal-oriented refinement and assignment towards architectural design. However, bridging the gap between the two research areas is still a critical issue [GE06] especially when quality requirements change. For a **mapping** between different aspects–in a similar way to the Softgoal Interdependency Graph–there are further approaches for example the quality models [MR04] and the Failure Mode Effect Analysis [FM49]. For a mapping based on impact analysis, Galster et al. [GE10] developed a method similar to our previous work [BR10].

3 Goal Solution Scheme

The Goal Solution Scheme GSS was developed to represent the mapping between elements of the problem space and those of the solution space during evolutionary software development. Its leading idea is inspired by the model-based design paradigm: to represent dependencies in an explicit way. The relationships between the elements of the scheme form a graph with

a structure similar to a tree. In the ideal case, the relationships between elements of different layers would be 1 to 1 relations; however in real situations the effects of scattering and tangling cannot be prevented completely. The layers (Fig. 1) correspond to stages of the development process and contain the elements of these stages. Each relationship between elements expresses a dependency: a change of one element requires changes of its related elements. The relationship's weight expresses the impact.

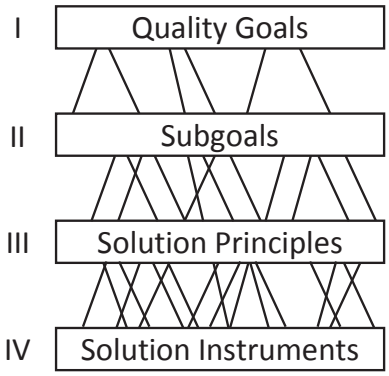

Figure 1: Layers of the Goal Solution Scheme

Layer I contains the top-level quality goals such as maintainability, evolvability, performance, portability, reliability, security, and usability. In layer II, there are the refined goals. The relationships of the transition between layers I and II represent the mapping of top-level quality goals to subgoals, similarly to a quality model.

Layer III contains solution principles from different research areas with a known impact on quality properties. Examples for software engineering principles are modularity and separation of concerns; examples regarding security are tamperproofness and verifiability.

Heuristics for problem solution can also be arranged on this layer. For reengineering, elements representing a violation of these principles are covered, such as design flaws, together with metrics for their detection. The transition from layer II to III represents the mapping of quality goals or subgoals to solution principles with an impact on the goal; and thus the transition between problem space and solution space. Regarding the QASAR method, the introduction of a functional solution for a so-called non-functional problem is represented by the dependencies of this transition. In the case of conflicting goals, resolutions can be found by an analysis of the dependencies of this transition—for example by the identification of potential trade-offs.

Layer IV covers solution instruments of two different categories with their relation to solution principles. Firstly, on this layer architectural solutions for forward engineering are provided. Patterns, styles, heuristics, frameworks, and components are examples for elements out of the architect's stock of solutions, sometimes called toolbox. Secondly, this layer provides reengineering activities for the removal of design flaws, such as architectural refactoring patterns. Each of the instruments of both categories is further described by (a) preconditions for applicability, and (b) a set of impact values regarding solution principles and quality goals. The relationships of the transition III - IV represent the impact of architectural solution instruments on the satisfaction of solution principles. These relationships are used for a classification solution instruments regarding their relation to quality goals and subgoals, as mentioned in section 4 in the second step.

4 Guidance for Goal-Oriented Architectural Design and Reengineering

Evolutionary software development demands for a combination of forward engineering and reengineering. The orientation on the goals is crucial for the success of both. The utilization of the Goal Solution Scheme for decision-making during architectural design is performed mainly in the same course for both, consisting of two steps:

Refinement and prioritization of quality goals regarding architecture. For competing quality goals, compromises or even scoping can hardly be performed at the layer of goals. The identification of conflicts and their resolution is much easier at the layer of subgoals or solution principles because trade-off effects at these layers facilitate a clear prioritization. Starting from the priorities of the goals, the impact values of the relationships between quality goals and subgoals or solution principles in the GSS are evaluated to calculate the weights. For example, if the goal maintainability got a higher priority than performance, the principle modularity and the design patterns *Façade* and *Abstract Factory* get a higher weight than the performance-related solution *Cache*. During this step the GSS is applied in a way very similar to the Softgoal Interdependency Graph mentioned in section 2.

Decision on solution instruments. The architectural decision-making—both for the development of the first, initial architectural design, or for the later iterations, or for reengineering—is performed in two parts, as discussed in earlier works [RW07]. Firstly, a preselection of the solution instruments is carried out to identify the applicable ones. For this purpose, the constraints of the design task are compared to the prerequisites of the solution instruments of layer IV. Second, the relationships of the transition II – III – IV of the GSS are evaluated. The impact values of the solution principles and architectural solution elements and the priorities of the goals are used to calculate weights for the preselected solution instruments to establish a ranking. The resulting ranked lists is presented to the architect as proposed solution instruments. This second step of the GSS goes beyond the concept of the Softgoal Interdependency Graph because the constraints of the technical solution are considered as a preselection.

The difference of reengineering decisions between those of pure forward engineering consists in the use of design flaws, bad smells and similar violations of solu-

tion principles instead of the solution principles themselves (layer III). For example, the design flaw Feature Tangling represents a violation of the solution principle Separation of Concerns. If this design flaw got a high weight during prioritization, the related metrics and interpretation rules are applied to determine affected parts of the system. The removal of the design flaws is performed by solution instruments (layer IV) such as refactorings. A decision for an appropriate refactoring is realized in the mentioned two steps: At first, the situation of each affected part is compared to the preconditions of the solution instruments for preselection; then the preselected refactorings are ranked regarding to their impact on the design flaw. The ranked list is proposed to the architect.

5 Establishment and Evaluation of the Scheme

The layers and transitions of the Goal Solution Scheme represent information from different sources which demand for differing ways of establishment and evaluation.

Literature review. A considerable number of solution elements together with rules and constraints for their applicability can be derived from literature, for example from catalogues of design patterns and architectural styles (layer IV). Knowledge about the refinement can be gathered from software quality standards and quality models (transition I – II). Various solution principles are covered by textbooks for the different disciplines of computer science (layer III). Examples for software engineering are design principles like encapsulation, modularity and separation of concerns. Furthermore, there are experience reports on the application of architectural solution elements with information on their impact on quality goals (transitions II – IV).

Acquisition of experiences from experts. Software architects can contribute additional solution elements for the further population of layer IV of the scheme. They can provide detailed information on the impact of the elements on quality goals to extend the relationships of the transition III - IV. Furthermore they can add information on constraints for the applicability of specific configurations and solution elements. Experienced architects are able to revise and improve the relationships and the weights within the GSS.

Iterative evaluation and improvement. The input to the GSS from the first two ways has to be considered as hypotheses because of the missing evaluation. To evaluate and to verify the hypotheses, the GSS has been applied in various case studies and projects, including the reengineering of the large robot software framework called Robot Software Ilmenau (RSI). The RSI project covered additional layer I quality goals like security and scalability. This addition as well as revisions during the evaluation affected the structure of the sub-goals (layer II) and the relationships including their weights. Furthermore, the revisions affected all transitions of the GSS. Moreover, a refinement of the methodical guidelines for the application of the GSS in both forward and reengineering occurred.

The application of the GSS in reengineering projects resulted in an addition of further reengineering strategies along with updated interpretation rules for the metrics for flaw detection and for the ranking of the reengineering activities. The additions occurred with two different characteristics: (1) Some new or updated reengineering strategies have been developed starting from (and driven by) quality goals. (2) New reengineering strategies have been developed, which were driven by specific design flaws and which were established during legacy code analysis.

During refinement and evaluation in the series of projects a decreasing rate of changes and updates to the GSS could be observed. This effect has been interpreted as an increasing level of maturity of the GSS. Beyond the evaluation in projects, empirical studies are planned to evaluate the content of the GSS regarding specific issues.

6 Tool support

Tool support is essential for the development of complex systems. Even for academic research, tools are a prerequisite for complex case studies. The presented concepts are applied in activities of architectural design and reengineering driven by quality goals. For these activities, a prototype tool is currently under development. It supports four main scenarios: **(1) Architectural design**: The developer enters prioritized quality goals (layer I), enters constraints, and obtains a ranked list of proposals for architectural styles and patterns (transition II – III – IV). **(2) Reengineering**: The developer enters prioritized quality goals, and obtains a ranked list of violations of principles i.e. design flaws (layer II). These flaws guide the next step of determination of reengineering activities (transition III – IV) and the related metrics. **(3) Extension of the toolbox** (layer IV): The architect adds a new architectural solution instrument with its impact on solution principles (transition III – IV) and possibly on quality goals and subgoals, and he is assisted by rearranging the classification of the toolbox by adjusting the set of weighted relationships. **(4) Revision of the relationships' impact factors of all transitions of the GSS:** An expert architect enters a set of prioritized quality goals and examines the resulting set of proposed solution instruments (layer IV). He selects the improper ones and adds missing ones, and the set of impact factors of the concerned relationships is adjusted accordingly. This tool is based on the repository *EMFTrace* which was developed for an explicit representation of dependencies between different models and artefacts of the development process, such as goal models, requirements specifications, as well as design models, configurations, and source code. The metamodel of the repository covers all relevant model elements of the considered models as well as the dependencies. It is based on the

Eclipse Modelling Framework EMF.

7 Conclusion and Future Work

In this paper, we presented the Goal Solution Scheme, which supports goal-oriented architectural design and reengineering. Following the principles of model-based design, it establishes an explicit mapping from quality goals and quality requirements to architectural solutions by dependencies. This mapping relates architectural solution elements like patterns, styles, frameworks and tools regarding their impact on quality goals. In the same way it relates reengineering and refactoring strategies to the quality goals. We have shown, how the Goal Solution Scheme facilitates tool support for scenarios of goal-oriented establishment and selection of architectural solutions. Similarly, tool support for reengineering activities for legacy analysis and refactoring is provided based on the Scheme. In this way the Goal Solution Scheme supports an integration of forward and reverse engineering activities for the sake of evolution. Furthermore, the establishment and refinement of the Goal Solution Scheme in industrial case studies and projects is explained, which results in maturation as well as in an evaluation.

For future works, the extension of the coverage is planned by incorporation of additional fields and their major quality goals, such as distributed systems, high performance systems, and highly reliable systems. As a continuous task, the extension of the stock of architectural instruments (layer IV) is performed by addition of pattern, refactorings, building blocks, tools, and products. Within this task, a continuous revision of the impact factors (transition II – IV) is planned by acquisition from a quality assessment of architectural models and whole systems. Furthermore a connection of the GSS concept with prediction approaches is considered, such as *Palladio* for performance prediction.

Acknowledgement

The research presented in this paper was conducted together with the PhD projects of Stephan Bode and Robert Brcina. It was partly funded by the Federal State Thuringia and the European Regional Development Fund ERDF through the Thüringer Aufbaubank with project no. 2007 FE 9041.

References

[Bo00] Bosch, J. Design and Use of Software Architectures. Addison Wesley, New York, 2000.

[BK02] Bass, L.J.; Klein, M.; Bachmann, F.: Quality Attribute Design Primitives and the Attribute Driven Design Method. In (F. van der Linden Ed.): Proc. Workshop Software Product-Family Engineering PFE2001, Springer, Berlin, 2002.; pp. 169-186.

[BR10] Bode, S., Riebisch, M.: Impact Evaluation for Quality-Oriented Architectural Decisions Regarding Evolvability. In Proc. ECSA2010. Springer, Berlin, 2010; pp. 182-197.

[CP09] Chung, L.; do Prado Leite, J.: On Non-Functional Requirements in Software Engineering. In (Borgida, A. et al. eds): Conceptual Modeling Foundations and Applications. Springer, Berlin, 2009, pp. 363–379.

[GE06] Galster, M.; Eberlein, A.; Moussavi, M.: Transition from Requirements to Architecture A Review and Future Perspective. In Proc. SNPD'06. IEEE, 2006. pp. 9-16.

[GE10] Galster, M.; Eberlein, A.; Moussavi, M.: Systematic selection of architectural styles. IET Softw. Oct 2010, Vol. 4, Issue 5, p.349–360

[FM49] Procedure for performing a failure mode effect and criticality analysis, MIL-P-1629, United States Military Procedure, 1949.

[HN00] Hofmeister, C.; Nord, R.; Soni, D.: Applied Software Architecture. Addison Wesley. New York, 2000.

[MR04] Marinescu, R.; Ratiu, D.: Quantifying the Quality of Object-Oriented Design The Factor-Strategy Model. Proc. WCRE 2004, IEEE, 2004, pp. 192-201.

[RW07] Riebisch, M.; Wohlfarth, S.: Introducing Impact Analysis for Architectural Decisions. Proc. ECBS2007, IEEE CS, 2007, pp. 381-390.

Presented at the conference Software Engineering 2011 (SE2011), Karlsruhe, Germany, 21 - 25 February, 2011