Towards Bridging the Gap between Goal-Oriented Requirements Engineering and Compositional Architecture Development

Sebastian Herold¹, Andreas Metzger², Andreas Rausch¹, Heiko Stallbaum²

¹ Clausthal University of Technology
Institute for Computer Science - Software Systems Engineering
Julius-Albert-Straße 4, 38678 Clausthal-Zellerfeld, Germany
{sebastian.herold, andreas.rausch}@tu-clausthal.de

² University of Duisburg-Essen, Campus Essen
Institute for Computer Science and Business Information Systems - Software Systems Engineering
Schützenbahn 70 45117 Essen, Germany
{andreas.metzger, heiko.stallbaum}@sse.uni-due.de

Abstract

Requirements engineering and architectural design are key activities for the successful development of software-intensive systems. Although both activities are strongly intertwined and interrelated, many steps to date are driven solely by the intuition and the architectural knowledge of individuals. Thus, systematic approaches are needed which could minimize the risks of wrong early requirements and architectural decisions and foster the explicit reuse of architectural knowledge especially for supporting early design decisions are. In this paper, we present our vision of supporting the early requirements and architectural decisions by making explicit the interactions between the early steps and artifacts in requirements engineering and architectural design and thereby reusing architectural knowledge. To this end, we propose to couple goal-oriented requirements engineering and compositional architecture development by means of a repository of reusable, generic architectural drivers.

1. Introduction

Requirements engineering (RE) and Architectural Design (AD) are essential for successfully developing high-quality software-intensive systems. RE and AD activities are intertwined and are iteratively performed. The architecture of the software system must satisfy its requirements, yet architectural constraints might prohibit certain requirements to be realized. This will then imply a change to the initial requirements (e.g., see [12, 5]). Further, additional requirements might be discovered during the development process, leading to changes in the architecture. Design decisions that are taken early in this iterative process are the most crucial ones, because they are very hard and costly to change later in the development process (e.g., see [13]).

Despite the above observations, the early activities in AD, like deciding on the coarse-grained architecture, are currently mostly guided by the intuition and the experience of the software architects. Therefore, those activities tend to be unsystematic, hard to predict or plan and difficult to perform by novices in architectural design. First solutions to this problem have been introduced e.g. by Chung et al. [3] and Baum et al. [2]. Chung et al. use high-level requirements artifacts (NFR models) to decide on the best architecture alternative. However, the proposed approach does not state how to actually derive an architecture alternative from the given requirements. On the contrary, Baum et al. show how to derive the architecture from the given requirements. Yet, the level of the requirements appears very detailed, thus making their approach appear to be better suited for the later development phases.

To improve on this situation, we propose to make the dependencies between high-level requirements artifacts and coarse-grained architecture alternatives explicit. In our envisioned approach, we plan to use and maintain an organization-wide (or potentially domain-wide) repository of generic architectural drivers for linking RE to AD.

From the RE perspective, the concrete set of high-level requirements of a system to be built contain concrete architectural drivers [1, 8], like “flexible sales system used”. Some of those concrete architectural drivers will be instantiations of the generic architectural drivers in the reuse repository. As an example, the above concrete architectural driver could
be initiated from the generic architectural driver “IT system supporting flexible business processes required”.

From the AD point of view, the generic architectural drivers are mapped to a repository of coarse-grained, reusable architectural components, which have proven to be solutions for realizing those generic architectural drivers. As an example, the usage of a service layer realizes the generic architectural driver mentioned above.

Once architectural drivers in the high-level requirements have been identified, these are mapped to one or more of the generic architectural drivers. From these generic architectural drivers, potential candidates for a coarse-grained architecture can be determined using the mapping from the repository of generic architectural drivers to the coarse-grained, reusable architectural components. The potential candidates are then evaluated concerning how well they actually satisfy the high-level requirements. This is done by following dependencies from the generic architectural drivers back to their instantiations in the requirements artifacts. Thus, our envisioned approach will support early decision making in AD by reusing architectural knowledge (cf. [9]) and relating it to high-level requirements artifacts.

The remainder of the paper will further detail this vision (Section 2) and will present an illustrative application scenario for the envisioned approach (Section 3).

2. The Approach

As introduced above, the basic idea of our approach is to derive coarse-grained architecture alternatives from high-level requirements by using a repository of generic architectural drivers. Figure 1 illustrates the relationships between RE and AD as well as the main kinds of artifacts that are used in our approach.

**Figure 1: Overview of envisioned approach**

Goal models (e.g., see [10, 11, 4, 14]) are used in RE to document the typically long-lived, strategic objectives that are aimed to be achieved by the system to be built. As such they represent one of the earliest requirements artifacts and provide the rationales for the detailed, solution-oriented requirements that are documented in the system requirements specification.

The main modeling elements that are found in goal models are goals, soft-goals, tasks, and the relationships between those elements. Goals model quantifiable intentions of the stakeholders that should be fulfilled by the system. Soft-goals are non-quantifiable intentions and as such essentially model non-functional (or quality) characteristics. The operationalization of goals can be modeled by tasks. Both tasks and goals may contribute to or hinder fulfilling the soft-goals.

When constructing and reasoning on the goal model, certain sub goals can be identified that are essential for achieving the overall goals and soft-goals of the system. Those sub goals – together with the relationships to the soft-goals they contribute to – constitute the concrete architectural drivers in our envisioned approach, i.e., concrete architectural drivers group goals and sub-goals that are most relevant for the design of the architecture (cf. [1, 8]). They massively influence the selection of architectural components to construct the system. For example, the goal of integrating an existing content management system (in order to contribute to the soft goal “intuitive access to repository of documents”) could be a concrete architectural driver for a system.

These concrete architectural drivers are matched to one or more of the generic architectural drivers in the repository. The aim of selecting appropriate generic architectural drivers is to select reusable architectural components, e.g. a certain distribution alternative, for a first sketch of one or more alternatives for a coarse-grained architecture of the system that supports the realization of the identified architectural drivers. Generic architectural drivers are generalizations (e.g., in the form of templates or parameterized models) of concrete architectural drivers from previous projects. Thus, the concrete architectural drivers identified in a project-specific goal model can be instances of generic architectural drivers. For example, the project-specific goal of integrating an existing content management system instantiates the generic architectural driver “Integration of third-party services desired”.

For each reusable architectural component, the experience is stored how well that component realized each of the generic architectural drivers. Additionally, the potential compositions of components are rated regarding how well they realize the generic...
Some compositions can realize a specific generic architectural driver better than others. For example, the composition of a service-oriented application layer and rich-client distribution would support the generic architectural driver “Integration of third-party services desired” better than an object-oriented application layer with same distribution.

By mapping generic architectural drivers to reusable components, we support a systematic approach to access and reuse architectural knowledge (cf. [9]). Thus, the software architects can select possibly several composed, coarse-grained architectures that will realize the architectural drivers in a proven way. The requirements engineers can evaluate these different alternatives of possible architectures with respect to the goal model. Their analysis leads to a set of profiles for the different architectures depicting how well they contribute to satisfying the individual goals and soft-goals. Such profiles can be computed from the goal models (e.g., see [5]) and are an important tool for decision support to decide on an optimal coarse-grained architecture.

Additionally, the feedback from AD to RE can lead to changes in the goal model. The different architecture alternatives may support generic architectural drivers for which no instantiation in the goal model has been modeled yet. Therefore, those additional generic architectural drivers can hint to goals and soft-goals that might have been overlooked in the initial goal model. Further, dependencies between goals and soft-goals might be uncovered in AD that in turn will uncover conflicts between the goals in the goal model (cf. [3, 11]). Thus, the choice of a coarse-grained architecture can drive further RE activities.

3. Application Scenario

To illustrate our approach, we describe an application scenario where the proposed integration of RE and AD is shown. We have chosen a business enterprise that is specialized in catalogue sales and is well established in its home market. To ensure further growth, the company wants to expand to new markets. Accordingly, the internationalization of catalogue sales is the most important goal in the recent business strategy of the company. Additional business goals are to keep personnel costs low. A first step towards internationalization is the reuse of IT systems in an international setting.

Figure 2 shows a goal model that documents the above goals of the enterprise using the Goal-oriented Requirements Language (GRL) [7]. It thus shows the results of early requirements engineering activities in our application scenario. The emphasis is put on the company’s internationalization goals, particularly with respect to effects on IT systems. Whenever a task contributes to the fulfillment of a business goal, this is modeled by a contribution link labeled with “+” or “−”. For example, the task “recruit multilingual personnel” has a negative impact on the soft-goal “minimization of personnel costs”, because the costs are expected to increase with the required skills of staff. Thus, in our scenario, the launch of local call centers seems to be an adequate decision, implying the need for a distributed sales system to support the processing of incoming orders by the call center staff. Optimizing this sales system for fast data input saves time and correspondingly personnel costs.

Additionally, the IT department of the business enterprise can access architectural knowledge from previous projects. This knowledge is available in form of generic architectural drivers and reusable architectural components, as shown in Figure 3. The figure shows different alternatives for the distribution architecture of a three layer information system and the possible combination with service-oriented vs. object-oriented application layer interfaces. Since the IT department has developed business supporting systems
in the past, software architects have identified several different generic architectural drivers like “IT system supporting flexible business processes required”. The repository of reusable architectural components contains information about how well the component combinations realized the stored generic architectural drivers. This is illustrated by the matrix in Figure 3, which shows whether the combination is a solution for the generic architectural driver (“+”) or not (“–”).

Also, infeasible combinations are documented as “n/a”.

When reasoning on the initial goal model with respect to architectural design, a central question arises: Does this model contain information that has an essential impact on the architecture of the resulting software system? To answer this question, our approach foresees to perform activities:

First, groups of goals and soft-goals are determined that constitute concrete architectural drivers. In our example, two architectural drivers can be identified:

- The goal “flexible sales system to be used” together with the soft-goal “adaptability of sales process” is a concrete architectural driver, as without realizing that goal, the soft-goal “adaptability of sales process” will not be achieved.
- The goal “sales system be optimized for fast input”, which is a sub goal of “run distributed sales system”, together with “low personnel costs” is a concrete architectural driver, as the soft-goal “low personnel costs” is only achieved when that goal is realized.

Second, the identified concrete architectural drivers are compared against the generic architectural drivers in the repository. In our example, two of those elements lead to generic architectural drivers:

- The goal “flexible sales system to be used” to achieve the soft-goal “adaptability of sales process” instantiates a generic architectural driver, described as “IT system supporting flexible business processes required” (GAD1 in Figure 3).
- The goal “sales system be optimized for fast input” as a sub goal of “run distributed sales system” leads to the generic architectural driver “fast data input in distributed IT system is required” (GAD2 in Figure 3).

Those two architectural drivers strongly affect the architecture. The first one can be realized by a service-oriented as well as by an object-oriented application layer. The second one limits the possible client realization to fat or rich clients, implying that the bulk of any data processing is executed on the client side. In contrast, a thin client only realizes the presentation layer and therefore the VPN bandwidth would slow down data processing.

By now we have identified components that satisfy best each of the relevant architectural drivers. However, some combinations of these components are pointless or even impossible. To get a set of proven combinations, we take a look at our repository of coarse-grained architectural components containing expert knowledge about combinations and constraints on possible combinations. In our example, the combination of a fat client with a service-oriented application layer doesn’t make any sense, because both presentation and service layer are client-sided. In contrast, the combination of a rich client with a service-oriented application layer makes sense and complies with the set of generic architectural drivers found. The alternative of combining a fat client with an object-oriented application layer also satisfies the two identified drivers.

In our application scenario so far, the initial goal model leads to two alternatives, a service-oriented rich client system (C2 in Figure 3) or an object-oriented fat client (C4 in Figure 3). This already is a valuable limitation to the space of possible architecture solutions. However, we can support further decision support in choosing between those two alternatives by investigating the component’s contribution to soft-goals that were not considered so far. In our scenario we see that the service-oriented solution satisfies the generic architectural driver “Future integration of third-parties services” (GAD3 in Figure 3). This information can be used to refine the goal model. For
example, Figure 4 shows an extension of the goal model of Figure 2, where the soft-goal “interoperability of IT systems” has been added. Based on this extended goal model, the architecture alternative that employs the service-oriented rich client (C2) should be preferred.

![Diagram of goal model]

Figure 4: Modified goal model based on input from architectural design

4. Conclusion

Requirements engineering (RE) and architectural design (AD) are key software development activities, which are strongly intertwined and interrelated. This paper has outlined our envisioned approach for making architectural knowledge explicit during the early stages of software development, considering the dependencies between RE and AD.

The main idea is to store generic architectural drivers in a reuse repository and to relate those generic architectural drivers to both requirements as well as reusable architectural components. Architectural knowledge about the relationships between requirements and proven architectural solutions is thus explicitly documented and can therefore be retrieved across individual development projects.

To realize the envisioned approach, a first objective that we pursue is to provide a precise definition and (partial) formalization of the relevant development artifacts (goals, generic architectural drivers, and reusable components) and their dependencies is needed. Based on that formalization, we plan to develop a technique, which includes guidelines and tool support. For tool support, this will include clarifying which activities in the overall approach can be automated. Especially during the identification of concrete architectural drivers and the composition of reusable components to architecture alternatives, we clearly see the limitations of such an automation and will thus focus on supporting the creative tasks of requirements engineers and architects. Tool support for searching relevant generic architectural drivers in the repository as well as the computation of profiles for architecture alternatives are more likely to be fully automated.

References


