Architecting Requirements

WenQian Liu

Department of Computer Science
University of Toronto
Toronto, ON, M5S 3G4, Canada

E-mail: wl@cs.utoronto.ca

Abstract

Producing software architectural design still poses many challenges in software engineering. I hypothesize that requirement structuring and refactoring provide means to identify subsystem decompositions as part of the architectural design. This paper outlines my doctoral research in validating the hypothesis and building refactoring methods based on it. Current progress and evaluation methods are described.

1 Problem Statement

Architectural design is largely creative work. Many good software architects rely extensively on their experience and domain knowledge. These factors not only make teaching difficult in the classroom setting, but also limit the number of skilled designers. Is it possible to transfer this artistic approach into the scientific realm? More specifically, I am interested in the following research questions.

- *Can we produce architectural designs systematically from the requirements in general, and how?*
- *Can we manage changes effectively as the software evolves during its lifetime, and how?*

Answers to these questions would bring many benefits. First, with a systematic approach, we can produce designs of more consistent quality, reduce human labor and error, train new designers effectively, and relate the design more closely to the requirements. Second, managing changes effectively can reduce cost and effort during maintenance. Third, a bridge between requirement phase and design phase can be established.

2 Prior Research

Active research in filling the gap between requirement engineering and architectural design has been pursued in recent years [1, 2]. One of them that shares a similar research goal is the work on GSE (Genetic Software Engineering) [11]. It provides a systematic approach in going from a set of functional requirements to a system-level design by using behavior trees for representation and integration. It focuses on the functional (or behavioral) requirements and emergent properties in design, but does not address quality attributes or tracing rationales.

Recognizing that tracing the rationale behind the design decisions is nearly impossible in many large system designs, Leveson [17] introduced the concept of intent specification based on research from the cognitive science community [24]. The focus of that work is to provide convenient notations and processes that match human cognitive capability at every step of system design. It uses intent to link the adjacent development phases such as the requirement phase – System Purpose level, and design phase – System Design Principles level, but does not provide systematic approaches for moving between levels.

Goal-oriented requirement specification techniques [25, 9, 23] are becoming a dominant approach. They provide not only notations but also processes for eliciting and specifying requirements. Recent work extends the approach towards agent-oriented design [26, 18, 22] and architecture prescription [4, 5].

Problem frames [15] capture generic patterns in problems. This allows software engineers to break down a large problem into recognizable subproblems and reuse their descriptions. However, the decomposition and composition of subproblems are not straightforward, and their connection to architectural design is even less obvious. Recent work has attempted to provide a solution in this direction [13, 19]. However, these extensions are geared towards a

pattern-oriented approach.

Although the prior researchers are all working towards a similar goal, their underlying hypotheses and principles vary. This gives many different approaches that share a similar general problem but solve different specific problems. In section 3, I will introduce my underlying hypothesis and describe the approach and method.

3 Proposed Research

In the requirement modelling phase, the emphasis is on the domain description, potential problems, and solution boundary. Little attention is paid to requirement structuring and refactoring. Traditionally, this analysis is implicitly done as part of the architectural design. In our new paradigm, it is separated from the other design activities.

I propose to place the aforementioned implicit analysis in a separate phase as part of the requirement realm (thus architecting requirements), so that the end products of requirement analysis have a structure that represents the logical view [16] of the system. This structure also enables incremental analysis of change requests at the requirement level before propagating to the design and implementation. *Architecting requirements* replaces the architectural design phase and fits in the development process between the requirement elicitation/modelling phase and the design/implementation phase as depicted in figure 1.

Figure 1. The new development process.

Hypothesis A good solution reflects the structure of the problem. This allows the solution to evolve in the same dimensions as the problem changes over time. Discovering problem structure will reveal properties of these solutions. Therefore, I propose to study the problem structure in search of good solutions.

There are two directions towards structuring requirements: from within the requirements and from the external environment perspective. In this light, I propose to structure and refactor requirements according to their life cycle and environment factors. A requirement life cycle describes the interactions the requirement has with others during the entire cycle of system development and operation. An environment factor is an aspect of the environment that affects the variability of the requirements. We use the requirement structure to derive the system decomposition and manage changes over time. The effectiveness of change management in this approach comes from the information integrity and organization, traceable rationale, and available analysis techniques over the structure.

3.1 Method

I introduce the *requirement hierarchy* method for architecting requirements. The constructs are described below. The entities of the hierarchy are *goals*, *needs*, *key needs*, and *capabilities*; the relations are *Reduce-To*, *refinement*, *composition*, and *interaction*. Each entity is a requirement. Figure 2 provides a structural view of the hierarchy.

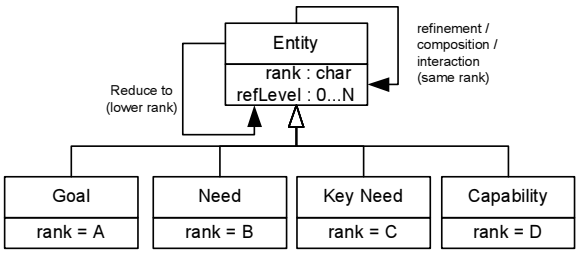

Figure 2. The class representation of the hierarchy.

There are four ranks, A (highest) through D (lowest), defined in the hierarchy, each corresponds to an entity type. The level of an entity in the hierarchy is defined in terms of the rank and the refinement level within the rank, refLevel where N is arbitrarily large. E.g. level A0, A1, B0, B1, etc. Entities of adjacent hierarchy ranks are linked by the *Reduce-To* relation. The entities of higher-rank serve as the intents of those below. Within the same rank, an entity can be elaborated in a new entity below through a *refinement* relation; an entity can be split into new entities or several entities joined into a new entity through the *composition* relation. These three types of relations are used to build the hierarchy. The ternary *interaction* relation is used in the lower two ranks to describe the interactions and interdependencies between entities of the same type, but acts like an entity and can be reduced to interactions in lower levels.

The information required to describe goals and needs can be extracted from early requirement models and documents created by using techniques such as *i** [25], KAOS [9], and Rational Unified Process (RUP) [16]. However, entities of the lower two ranks in the hierarchy are to be constructed using two new refactoring techniques, *life-cycle analysis* and *environment factor analysis*.

A top-down construction of the requirement hierarchy is outlined and the refactoring techniques are described below. The Reduce-To relation is used between the steps. Requirements of a Workplace Content Management software [14] are used as a running example.

1. Encode the visions and business opportunities in *goals*. Each goal should be stated as independently of others as possible, yet collectively, they present the rationale of why the new system should be built.

   This is a more narrow definition of *goal* than those from [9]. Here, a goal only encapsulates the business context and rationale, thus is a business goal. But in [9], a goal refers to any objective that the composite system should fulfill, both technical and business.

   Example

   Goal: Improve profitability by decreasing costs.

2. Identify the problems that may hinder the achievement of goals and encode in *needs*. A need describes either the negative impact or deficiencies of a phenomenon or observable improvement and mitigation of the impact, and how the stakeholders are affected. Refinements of a need can be its solutions or new problems.

   Example

   Need: Hard-copy documentation incurs a high cost. How can we reduce the cost while maintaining the accessibility and control of the documentation?

   Refined need: Store documents digitally and provide controlled online access. When access demand drops, change the storage to low cost media.

3. Use the *life-cycle analysis* to separate any independent concerns within a need into new needs, and join those that share common concerns. The refactored needs are called *key needs*. Examples are omitted thereafter due to limited space.

   The *life-cycle analysis* comprises the following steps:

   (a) Define the temporal coupling of the associated events occurring during the lifespan of the requirements (needs) including development and maintenance phases.

   (b) Use the coupling to identify differences and similarities between requirements.

   (c) Use the difference and/or similarity to decide splitting a need or joining one with another.

   This analysis reduces the information scattering and provides an overall integrity to the requirements (key needs). So far, we have not considered the external constraints or how they might affect the system. The next step will add the missing piece.

4. Derive the *capabilities* by refactoring the key needs according to the environment factors including quality attributes, prioritization, risks, external policies, technological impact, and the conditions of the physical computing environment wherein the system operates.

   In the *environment factor analysis*, orthogonal dimensions are defined based on the factors to represent the environment, and regions are set up with respect to the values in each dimension. Requirements are evaluated in each dimension with those that cross multiple regions subject to separation, and those that fall into the same region subject to consolidation.

   Capabilities represent the changing of perspective from *what is wanted* to *what is to be provided* by the system. They are the components of the system decomposition and can be considered as the high-level features.

Working in the requirement space, designers can deduce the subsystem decomposition from the requirement hierarchy. This provides a systematic approach in going from the requirements to the architectural design. It allows change requests to be iteratively built into the hierarchy which helps engineers to analyze the overall impact to the requirement structure and design before implementing the changes.

4 Current Progress and Expected Contribution

Although it is a work-in-progress, the requirement hierarchy method is being tested on a number of IT projects at the IBM Toronto Research Center [21]. We have built the top-rank levels of the hierarchy for selected projects and find this approach helpful in clarifying problems and gaining understanding of the requirements. The development of the analysis techniques is ongoing and we are looking for an entity specification language that permits tool support for automatic analysis. The change management is achieved by analyzing the impact of change to the requirement hierarchy incrementally with tool assistance.

To reduce manual work, tool support is mandatory. The following lists the key features for the tool.

- Provide notations and graphical user interface to support the construction of the requirement hierarchy.
- Store the hierarchy using flexible internal representation that allow customizable view and manipulation.
- Provide automated assistance in the discovery of temporal coupling of events and suggestion of join and split actions for the *life-cycle analysis*. The designer makes the final decision of the refactoring choices but is saved from tedious routine work.
- Provide predefined environment dimensions and regions with machine-assisted mapping from requirements to regions in the *environment factor analysis*.

- Automatically generate graphical and textual documentation for requirements and architectural design.

This work uses hierarchies to structure requirements and provides analysis techniques for designers to refactor requirements in order to identify the right form. This not only gives rise to the system decomposition but also provides a foundation for further design and change management.

5 Evaluation

The evaluation of the proposed research has two parts.

Part A On the validity of the hypothesis.

Identify classes of problems that exhibit the characteristics of the hypothesis. Start with the available projects in IBM [21] and exemplars from the literature [3, 8, 12], apply the proposed method, evaluate the results against criteria designed based on those in [7, 20, 10, 6], and summarize and generalize the characteristics of the problems to form a classification.

Validate the classification on industrial case studies.

Part B On the effectiveness of the proposed method.

Examine how well the problem structure, defined by applying the method, assists in finding good solutions. Evaluate both the strength and the weakness of the method in discovering and building the structure.

Summarize and classify the characteristics of the problems where the method is effective or ineffective.

References

[1] In J. Castro and J. Kramer, editors, *"The First International Workshop on From Software Requirements to Architectures (STRAW'01)"*. At the 23rd International Conference on Software Engineering, Toronto, 2001.

[2] In D. M. Berry, R. Kazman, and R. Wieringa, editors, *"The Second International Software Requirements to Architectures Workshop (STRAW'03)"*. At the 25rd International Conference on Software Engineering, Portland, 2003.

[3] L. Bass, P. Clements, and R. Kazman. *"Software Architecture in Practice" (2nd Ed.)*. Addison-Wesley, 2003.

[4] M. Brandozzi and D. E. Perry. "From Goal-Oriented Requirements to Architectural Prescriptions: The Preskriptor Process". pages 107–113. In [2].

[5] M. Brandozzi and D. E. Perry. "Transforming Goal Oriented Requirement Specifications into Architecture Prescriptions". pages 54–60. In [1].

[6] J. Cameron, A. Campbell, and P. Ward. "Comparing Software development Methods: An Example". *Information and Software Technology*, 33(6):386–402, 1991.

[7] P. Clements, R. Kazman, and M. Klein. *"Evaluating Software Architectures"*. Addison-Wesley, 2001.

[8] A. Dardeene, A. van Lamsweerde, and S. Fickas. "Goal-Directed Requirements Acquisition". *Science of Computer Programming*, 20:3–50, 1993.

[9] A. Dardenne, A. van Lamsweerde, and S. Fickas. "Goal-Directed Requirements Acquisition". *Science of Computer Programming*, 20:3–50, 1993.

[10] A. Davis. A comparison of techniques for the specification of external system behavior. *Comm. ACM*, 31(9):1098–1115, 1988.

[11] R. Dromey. "Architecture as an Emergent Property of Requirements Integration". pages 77–84. In [2].

[12] M. Feather, S. Fickas, A. Finkelstein, and A. v. Lamsweerde. "Requirements and Specification Exemplars". *Automated Software Engineering*, 4(4):419–438, 1997.

[13] J. G. Hall, M. Jackson, R. C. Laney, B. Nuseibeh, and L. Rapanotti. "Relating Software Requirements and Architectures using Problem Frames". In *Proceedings of IEEE International Requirements Engineering Conference (RE'02), Essen, Germany, 9-13 September 2002*, 2002.

[14] IBM. "Knowledge, expertise and content management software". `http://www-306.ibm.com/software/lotus/knowledge`, 2004.

[15] M. Jackson. *"Problem Frames: Analysing and Structuring Software Development Problems"*. Addison-Wesley, 2001.

[16] P. Kruchten. *"The Rational Unified Process An Introduction Second Edition"*. Addison Wesley, 2000.

[17] N. Leveson. "Intent Specifications: An Approach to Building Human-Centered Specifications". *IEEE Transactions on Software Engineering*.

[18] J. Mylopoulos et al. "Tropos - Requirements-Driven Development for Agent Software". `http://www.troposproject.org/`, 2004.

[19] L. Rapanotti, J. G. Hall, M. Jackson, and B. Nuseibeh. "Architecture-driven Problem Decomposition". To appear.

[20] S. Sigfried. *"Understanding Object-oriented Software Engineering"*. IEEE Press, New York, 1996.

[21] M. Starkey and S. Lymer. Personal communication. `http://www.can.ibm.com/torontolab/`, Feb 2004.

[22] A. van Lamsweerde. "From System Goals to Software Architecture". In M. Bernardo and P. Inverardi, editors, *Formal Methods for Software Architectures*, pages 25–43, 2003.

[23] A. van Lamsweerde, R. Darimont, and E. Letier. "Managing Conflicts in Goal-Driven Requirements Engineering". *IEEE Transactions on Software Engineering*, 24(11):908–926, 1998.

[24] K. J. Vicente. *Cognitive Work Analysis*. LEA, 1999.

[25] E. Yu. *" Modelling Strategic Relationships for Process Reengineering (Agents, Goals)"*. PhD thesis, University of Toronto (Canada), Department of Computer Science, 1995.

[26] E. Yu et al. "Goal-oriented Requirement Language (GRL)". `http://www.cs.toronto.edu/km/GRL/`, 2004.