Patterns and their impact on system concerns

Michael Weiss
Department of Systems and Computer Engineering
Carleton University, Ottawa, Canada
weiss@sce.carleton.ca

Abstract

Making the link between architectural decisions and system concerns explicit is a major contribution that patterns can make. Over the past decade, there have been several efforts to close the gap between requirements and architecture by using patterns. In this paper, our goal is to take a step back and survey these different contributions, as well as related efforts in other communities (such as the work on aspect-oriented requirements engineering). From these, we identify common elements and present a perspective on how to move forward. This thematic track on Pragmatic and Systematic Approaches in Applying Patterns should provide a good conduit for this discussion.

1 Introduction

There has been much recent interest in understanding the link between patterns and system concerns, also known as non-functional requirements. There is a well-recognized gap between requirements and architecture. We also know that system concerns may be satisfied to a differing extent by alternative architectures, and that we need to explore and evaluate architectural alternatives (Grau and Franch 2007). The system architect is faced with designing a system that meets both functional and non-functional requirements.

Harrison and Avgeriou (2007) suggest that patterns are a good way to understand the impact of architectural decisions, because they contain information about consequences and context of the pattern usage. However, they also go on to state that this information has been of limited use, because it is not presented consistently or systematically at present. They propose to integrate the information about the impact of patterns on system concerns in order to increase the usefulness of architectural patterns.

Over the past decade, a number of research groups have made contributions to our understanding of the link between patterns and system concerns. However, their work has been dispersed and we have not leveraged the results as well as we could have. As a step towards advancing these efforts, our goal is to summarize the existing research on the problem and to identify lessons learned and questions for future research.

We have divided the surveyed contributions into three streams. The first stream is on work that explicitly aims to link patterns and system concerns. Much of this work has been carried out with the goal of supporting the selection of patterns, our second stream. The third stream is concerned with work on documenting the rationale for architectural decisions and trade-offs. Here, we will only review some representative examples.

2 Patterns and system concerns

Several papers are concerned with making an explicit link between patterns and system concerns.[1] There are three perspectives within this stream: non-functional requirements modeling (Gross and Yu 2001; Araujo and Weiss 2002; Chung et al. 2002; Mussbacher, Amyot and Weiss 2006), layered system architecture and non-functional patterns (Fernandez 2003), and effective information organization (Harrison and Avgeriou 2007).

Gross and Yu (2001) examine the applicability of the well-established Non-Functional Requirements (NFR) framework by Chung et al. (2000) to the representation and application of patterns. The NFR framework makes the relationship between non-functional requirements and design decisions explicit. Gross and Yu extract the contributions of a pattern on non-functional requirements from a textual analysis of the problem statement. They then model the impact of a pattern in terms of "softgoals". Softgoal is the term used by Chung et al. (2000) to indicate that, unlike functional requirements, non-functional requirements cannot be achieved in an absolute sense, but only to some degree. Gross and Yu (2001) use softgoals to represent the forces that a pattern helps achieve or prevents from achieving. Solutions of patterns are represented as operationalizing goals. They are said to "operationalize" goals, as they turn those goals into solutions that help achieve those goals in a specific manner. Side effects of a solution are also made explicit as part of their analysis. This approach allows the comparison and consequent selection of patterns in terms of their impact on system concerns.

Araujo and Weiss (2002) improve on the work by Gross and Yu (2001) in an effort to create a catalog of the impact of patterns on system concerns using a consistent vocabulary of forces for a given domain (their domain is distributed system design). They show how patterns can be mapped to architectural issues and decisions, resources, constraints, and system concerns. Like Gross and Yu (2001), they model patterns using softgoal graphs. A link between a pattern and a force in the goal graph (which the authors call a "force hierarchy") indicates that the pattern contributes to its achievement. Each pattern is the result of a trade-off or balance between forces. Representing the contributions of a pattern as a softgoal graph makes the contributions of the pattern toward achieving the domain forces explicit. It highlights the trade-offs made by a

[1] I thank my shepherd for pointing out that these patterns are also, in some sense, about the selection of patterns. Representing the impact on system concerns is a precondition for selecting patterns. For example, when a security pattern mitigates a particular security threat, this pattern becomes a candidate to be selected when this threat is faced. However, none of the papers in this section directly discusses the application to selection. Yet, clearly pattern selection builds on pattern representations such as those developed here.

pattern. For example, it may achieve certain forces, but hinders the achievement of other forces. It also makes visible forces that remain unresolved after applying a pattern.

Chung et al. (2002) document the rationale for selecting design patterns that are used together (something they call a "pattern set") using softgoal graphs. Their approach marries goal-oriented modeling with design reuse in the form of patterns. The approach is also based on the NFR framework. It proposes to model the functional and non-functional requirements of a system using the NFR framework, refine and prioritize them, and establish architectural alternatives that meet these requirements. Next, a system designer should consider patterns that satisfy these architectural alternatives, and analyze the trade-offs among the architectural alternatives and their associated patterns. The approach ends with the selection of architectures and patterns that best satisfy the non-functional requirements identified, and instantiating the patterns in the design. For example, indirect and direct invocation are two architectural alternatives to notify subscribers, and the Observer pattern is a way of implementing the indirect invocation style. Indirect invocation leads to a loosely coupled system, which improves maintainability. This link is modeled through contributions. Pattern dependencies are also accounted for in this approach, so selecting an Observer pattern would imply using a Factory pattern.

System concerns are impacted at all levels of a system, as pointed out by Fernandez (2006). His particular focus is on security: access control and authorization constraints defined at the application level need to be enforced by lower levels, such as database, distribution, and hardware levels. Patterns provide a systematic way of reusing design knowledge to build systems that meet specific non-functional requirements. Extending the proposal of Araujo and Weiss (2002), Fernandez' approach also incorporates the notion of mapping between patterns at different levels of abstraction:

> We can define patterns at all levels. This allows a designer to make sure that all levels are secured, and also makes easier propagating down the high-level constraints.

For example, the implementation of the Authorization pattern at the application level requires the use of the Single Access Point and Check Point patterns at the system level, as well as patterns for file access and process creation at the operating system level.

Later, Mussbacher, Amyot and Weiss (2006) more clearly distinguish between a force and a non-functional requirement than earlier work. They formalize architectural patterns with the Goal-oriented Requirements Language (GRL). Forces and contributions of individual patterns are captured using GRL. Combinations and side effects (correlations) are described with AND graphs, and alternative combinations for a given (functional) goal are represented with an OR graph. With the help of strategies (that is, initial selections of candidate patterns) and propagation rules, designers can assess the impact of their selection on the forces and find a suitable solution in their context. This context can itself be modeled with GRL, first at the actor/dependency level and then at the level of intentional elements (goals, softgoals, tasks, etc.) for the system. This enables global and rigorous assessments to be made, even when many functional subgoals are considered.

Harrison and Avgeriou (2007) analyze the impact of patterns on system concerns and propose a way of organizing this information so that it is more accessible and informative. They selected well-known architectural patterns and documented the consequences of applying these patterns in terms of their strengths and liabilities in the form of tables that allow for easy comparison. Commenting on their analysis, they remark that using patterns makes it less likely that architects overlook important consequences of architectural decisions. In their words, this "relieves the architect of the burden of being expert in all the quality attributes". In comparison to other methods that center around system concerns such as QASAR (Bosch 2000), patterns focus more on the interaction among patterns and quality attributes than on specific system concerns.

Table 1 compares these approaches in terms of their features.

3 Selection of patterns

Other approaches also target the selection of patterns, and are, thus, presented in this section, although they all include a representation of the system concerns impacted by a pattern. This stream includes work on pattern-based design (Weiss 2003), design space visualization (Zdun 2006), architectural decision trees (Fernandez et al. 2006), decision-theoretic approaches to automate pattern selection (MacPhail and Deugo 2001), and pattern search engines (Weiss and Mouratidis, 2008). Note that we limited our attention to approaches that use system concerns as part of their decision process. There are other approaches to pattern selection that do not consider system concerns.

Weiss (2003) describes a pattern-based approach to system design that is both goal-driven (top-down) and pattern-driven (bottom-up) as in **Error! Reference source not found.**. Their approach involves five steps: identify domain forces, document roles (patterns are documented as role diagrams in this approach), document patterns and their dependencies, identify the overall design goals (expressed in terms of the forces implied by the requirements), and select patterns that help achieve them. The last step is concerned with selecting patterns. The first three steps are steps that only pattern writers go through, whereas the last two steps are performed by designers, who want to apply the patterns.

Having identified the overall prioritized design goals, the architect should now select the patterns that help achieve them. As in Araujo and Weiss (2002), the approach relies on a softgoal representation of the patterns. The selection is performed manually with the help of a reverse index that lists the patterns achieving a particular force. This index can be derived from the individual softgoal graph model of each pattern. Weiss (2003) also remarks that if we want to evaluate the effect of applying several patterns, we can combine the softgoal graphs for the individual patterns, and obtain a softgoal graph in which the patterns are operationalizations (designs or implementations that achieve the softgoals). We can also compare the results of applying alternative solutions to the same problem suggested by different patterns. The choice of the pattern depends on the prioritization of the forces by the designer (that is, there is no single best solution).

	Gross and Yu (2001)	Araujo and Weiss (2002)	Chung et al. (2003)	Mussbacher et al. (2006)	Zdun (2007)	Harrison and Avgeriou (2007)
Forces	Softgoals	Softgoals	Softgoals	Softgoals	Criteria	Columns
Functional goals	Goals			Goals	(Questions?)	
Patterns	Operationali-zations	Operationali-zations	Operationali-zations	Tasks	Options	Rows
Pattern dependencies			Contributions	Task decomposition, actor dependencies	Follow-up questions	
Force relationships	Contributions	Contributions	Contributions	Goal graph with stakeholders		
Context		Goal graph	Priorities	Actor dependencies		

Table 1. Features of the different pattern representations

Fernandez et al. (2006) propose the use of architectural decision trees to record selected patterns as well as alternatives that were considered but discarded. A decision tree allows architects to make decisions about system concerns vs. functional decisions. Architects can also later backtrack in the tree and make different decisions as the outcome of a decision was not the expected one or the requirements change.

Zdun (2007) describes an approach to reduce the complexity of pattern selection by employing pattern language grammars and design spaces. The approach considers quality goals (which the author equates with forces) and pattern variants. The design space approach extends the question-option-criteria (QOC) notation from HCI, which is related to the goal-question-metric approach from software engineering. Instead of using QOC analysis to visualize alternative design decisions, Zdun (2007) applies it to document the impact of alternative patterns to the quality attributes in forces and consequences. As in the work of Gross and Yu (2001) and Araujo and Weiss (2001), the level of abstraction is, therefore, that of patterns, not that of concrete design decisions. The design space approach is recursively applied, if related patterns raise new design questions.

Some proposals have been made to automate the selection of patterns. For example, McPhail and Deugo (2001) use a weighted distance metric (where each force is weighted by its priority) to search for matching patterns among a large number of patterns. An interesting aspect of their proposal is to decompose forces (such as performance and maintainability) into object-oriented quality metrics. The level of satisfaction of a force can thus be automatically computed from the object model of the pattern solution. Their approach is particularly suitable to compare variants of a pattern, that is, to determine which of various versions of, say, the Visitor pattern is best for a particular design.

Schumacher (2003) describes an expert system for the retrieval of security patterns. He proposes a representation of meta-information for security patterns, which includes the standard context, problem, solution elements as well as pattern dependencies, but also security-specific elements such as information about the threats a pattern protects against. Through a set of inference rules that encode knowledge about the pattern elements and pattern relationships, the expert system supports navigation of patterns based on pattern relationships, and detection of conflicts and comparison of alternatives. There is also some support for the qualitative comparison of patterns in terms of non-security forces.

Current work by Weiss and Mouratidis (2008) proposes a search engine for patterns that employs the pattern representation by Mussbacher, Amyot and Weiss (2006). Patterns are represented in terms of their impact on system concerns. A rules engine is used to reason about the effect of combining patterns on system concerns, and to identify trade-offs between system concerns. Its input is a set of system concerns that need to be satisfied, and its output a set of patterns that meets all requirements, if they can be satisfied, or most of them. The search engine can produce multiple pattern sets, ranked on how they satisfy the input requirements. The reasoning process also considers pattern dependencies: one important implication is that each pattern may add new requirements of its own, which then drive the selection of further patterns.

4 Rationale for architectural decisions

This stream is concerned with related work on documenting the rationale for making architectural decisions. It also looks at efforts undertaken under the umbrella of separation of concerns. There are two groups of papers reviewed here: the work by Akerman et al. (2006), Zimmermann et al. (2007) and Brito et al. (2007), which models architectural decision making in terms of reasoning about system concerns, but does not make explicit use of patterns, and work that treats patterns as reusable architectural knowledge (Zimmermann et al. 2008; Harrison and Avgeriou 2007). The former work is included here, because it has direct bearing on how we can reason about the impact of patterns on system concerns, if we treat patterns as architecture knowledge.

Akerman et al. (2006) propose an approach to software development that focuses on architectural decisions and uses an ontology to capture the architecture. The ontology has major components for capturing stakeholder concerns, architectural assets, architectural decisions, and a transformation roadmap. They present detailed models of these components, which could provide the basis for a common vocabulary for reasoning about architectural decisions. According to the authors, a pattern catalog of the type described in (Araujo and Weiss 2002) may be a start to populate an enterprise architecture ontology. Recent work by Zimmermann et al. (2007) on an Architectural Decision Knowledge Wiki applies the theoretical framework Akerman et al. (2006) and implements it in a tool. This work considers three levels of architectural decisions: concept, technology, and asset. Concepts are patterns or abstract principles.

Zimmermann et al. (2008) combines pattern languages and architectural decision models. The proposed ArchPad method facilitates the selection of patterns and provides traceability from generic patterns to project-specific adaptations of those patterns. Patterns are treated as a source of reusable architectural knowledge, whereas architectural decision models document specific design decisions and the alternatives considered. Applying a pattern means to make an architectural decision; to address the consequences of a pattern, further architectural decisions need to be made.

The impact of architectural decisions on system concerns is also heavily researched in the aspect-oriented requirements engineering community. A recent example is Brito et al. (2007), who propose to use the Analytic Hierarchy Process to resolve conflicts between system concerns. Given a set of alternatives and a set of decision criteria, the method will determine the best alternative in a rigorous manner.

Quality attributes often interact. Changes to a system that improve one set of quality attributes usually have unforeseen side effects on quality attributes elsewhere, as noted by Harrison and Avgeriou (2007). An example of the complexity of the interaction of non-functional requirements has been documented in Dyson and Longshaw (2004).

The Non-Functional Requirements (NFR) framework in Chung et al. (2000) is a goal-oriented approach for modeling interactions between NFRs, and deriving a “good” or (with respect to the user’s priorities) optimal software architecture. It introduces the

notion of a softgoal. The prefix "soft" indicates that softgoals are often subjective in nature, unlike functional (or "hard") goals. The NFR framework is used for documenting design rationale, and it helps represent the relationships between design decisions and non-functional requirements. Its extension within the Goal-oriented Requirements Language (GRL) can also model the viewpoints of multiple stakeholders (GRL 2007).

5 Lessons Learned

Our first set of lessons learned from our survey of the literature indicates that the literature on patterns and system concerns is still fragmented:

- There are several dispersed research efforts on enhancing our understanding of how to link patterns and system concerns
- These efforts lack a common vocabulary and do not agree on notation[2]
- There is also a lack of large case studies to validate the proposed approaches, specifically ones with industrial involvement

On the other hand, as this paper hopes to show, there are many common ideas underlying these approaches, and their synergy should be better exploited:

- Patterns make the communication of architectural decisions easier
- Architectural decisions are made in terms of system concerns: solutions to the same functional requirements differ in their impact on NFRs
- Patterns capture reusable architectural knowledge, so use of patterns can reduce the effort on documenting architectural decisions and help capture rationale
- There are several related notions to represent the concept of force in patterns, and there is an important distinction between force and non-functional requirement
- Pattern selection must take pattern dependencies into account (different approaches use goal decomposition and pattern language grammars)
- While forces are often treated as one-dimensional (as in "performance" is a force), they often interact in rich and complex ways
- Not all notations make the context in which a pattern is applied explicit

Acknowledgement

My thanks go to my shepherd Ed Fernandez whose probing questions and insights have helped me clarify my initial ideas.

[2] This is not to say that a variety of notations is bad, but it may be indicative of a fragmentation of the literature into different "closed" schools

References

Primary references are indicated with a (*). The other references are provided as sources supporting the argument in the paper, but are not essential reading.[3]

* Akerman, A., and Tyree, J., Using Ontology to Support Development of Software Architectures, IBM Systems Journal, 45(4), 813-825, 2006

* Araujo, I., and Weiss, M., Linking Non-Functional Requirements and Patterns, Conference on Pattern Languages of Programs (PLoP), 2002

Bass, L., Clements, P., and Kazman, R., Software Architecture in Practice, Addison Wesley, 2003

Bosch, J., Design and Use of Software Architecture- Adopting and Evolving a Product-Line Approach, Addison Wesley, 2000

Brito, I., Viera, F., Moreira, A., and Ribiero, R., Handling Conflicts in Aspectual Requirements Compositions, Transactions on Aspect-Oriented Software Design III, LNCS 4620, 144–166, 2007

Chung, L., Nixon, B., Yu, E., and Mylopoulos, J., Non-Functional Requirements in Software Engineering, Kluwer, 2000

* Chung, L., Supakkul, S., and Yu, A., Good Software Architecting: Goals, Objects, and Patterns, Information, Computing & Communication Technology Symposium, 2002

Davidsson, P., Johansson, S., and Svahnberg, M., Using the Analytic Hierarchy Process for Evaluating MAS Architecture Candidates, International Workshop on Agent Oriented Software Engineering, 2005

Dyson, P., and Longshaw, A., Architecting Enterprise Solutions, Wiley, 2004, pp. 18-22 discuss balancing non-functional requirements

* Fernandez, E., Security Patterns, International Symposium on System and Information Security, Keynote, 2006

Fernandez, E., Cholmondeley, P., and Zimmermann, O., Extending a Secure System Development Methodology to SOA, 2006

[3] In this way, I hope to balance the trade-off between the expectation that a pattern paper should only include a small number of references, and acknowledging the large number of sources that have inspired and shaped this paper.

Grau, G., and Franch, X., A Goal-Oriented Approach for the Generation and Evaluation of Alternative Architectures, European Conference on Software Architecture, LNCS 4758, Springer, 139-155, 2007

* Gross, D., and Yu, E., From Non-Functional Requirements to Design through Patterns, Requirements Engineering, 6(1), 18–36, 2001

GRL, http://www.cs.toronto.edu/km/GRL, last accessed in March 2007

* Harrison, N., and Avgeriou, P., Leveraging Architecture Patterns to Satisfy Quality Attributes, European Conference on Software Architecture, LNCS 4758, Springer, 263-270, 2007

McPhail, J.C., and Deugo, D., Deciding on a Pattern, International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, LNCS 2070, 901–910. Springer, 2001

* Mussbacher, G., Amyot, D., and Weiss, M., Formalizing Architectural Patterns with the Goal-Oriented Requirement Language, Nordic Pattern Languages of Programs Conference (VikingPLoP), 2006

* Schumacher, M., Security Engineering with Patterns, LNCS 2754, Springer, 2003

Zimmermann, O., Gschwind, T., Küster, J., Leymann, F., and Schuster, N., Reusable Architectural Decision Models for Enterprise Application Development, International Conference on Software Architecture, LNCS 4880, 15-32, Springer, 2007

* Weiss, M., Pattern-Driven Design of Agent Systems: Approach and Case Study, International Conference on Advanced Information Systems Engineering, LNCS 2681, 711-723, Springer, 2003

Weiss, M., and Mouratidis, H., Selecting Security Patterns that Fulfill Security Requirements, International Conference on Requirements Engineering, 2008

* Zdun, U., Systematic Pattern Selection Using Pattern Language Grammars and Design Space Analysis, Software Practice and Experience, 27, 983-1016, 2007

Zimmermann, O., Zdun, U., Gschwind, T., and Leymann, F., Combining Pattern Languages and Reusable Architectural Decision Models into a Comprehensive and Comprehensible Design Method, Working IEEE/IFIP Conference on Software Architecture, 157-166, 2008