A SYSTEMATIC ANALYSIS PATTERNS SPECIFICATION

R. Raminhos, M. Pantoquilho
UNINOVA – Desenvolvimento de Novas Tecnologias,
2829-516 Caparica, Portugal

J. Araújo, A. Moreira
CITI/FCT, Universidade Nova de Lisboa
2829-516 Caparica, Portugal

Keywords: Analysis patterns.

Abstract: Analysis Patterns are indicative analysis solutions for a recurrent problem. Many patterns have been proposed and are successfully used. The writing of a pattern follows a specific structure that can be tailored to each author's needs. We have developed an analysis pattern template that solves some previously identified gaps on other approaches. This paper focuses on the definition of a systematic process to guide developers to fill in that analysis pattern template. This process will contribute to the unification of the analysis patterns representation, and thus for their understandability and completeness.

1 INTRODUCTION

Patterns are a well-known and broadly used technique to specify software design and implementation (Gamma et al., 1995). Analysis patterns usage is fast growing in the software engineering community. Many templates to specify analysis patterns have been proposed (e.g. (Fowler, 1997), (Fernandez and Yuan, 2000), (Konrad and Cheng, 2002) and (Robertson, 1996)). As expected, these templates are based on the work already available for design patterns, and are tailored according to each author's needs and style. This led to a wide variety of analysis patterns styles, which compromise their usability by increasing the difficulty in analyzing and understanding different pattern templates.

In an attempt to unify the existing analysis patterns templates, we propose a template that combines the common features of the existing ones and adds new features that were missing. Since this template provides a wide variety of information, filling in all the fields may be a difficult task. So, we also propose a systematic process to assist software developers in building analysis patterns.

This paper is organized as follows. Section 2 describes the proposed template. Section 3 shows our process model. Section 4 discusses some related work. Finally, Section 5 draws some conclusion and points out directions for future work.

2 AN ANALYSIS PATTERN TEMPLATE

Currently, the specification of analysis and requirements patterns lacks key information for its usage by both young and experienced developers. Including certain specific details in the descriptions of patterns would facilitate the developer's work, as this would help them to take the right decisions on how to use the patterns efficiently and successfully. Examples of such information are a detailed list of functional and non-functional requirements, dependencies, conflict management, static and dynamic models, related patterns and anti-patterns.

Moreover, each existing approach suggests a different template. Also, guidance on how patterns should be specified is not clearly defined. The lack of consensus on this matter, therefore, prevents those approaches from being accepted widely.

We propose a template that unifies the existing ones and defines new entries to fill the identified gaps (Figure 1). An initial attempt to handle this problem was made in (Pantoquilho et al., 2003). These entries provide detailed information that covers from requirements descriptions and structural and behavioural modelling of the pattern to evolution issues, which are essential for the precise application of the pattern by the developer.

1. **Name:** Pattern identifier.
2. **Also known as:** Additional names that can also identify this pattern.
3. **History:** Chronological register of all previous versions of this pattern. The following notation should be used: {Date, Author, Reason and Changes}. To be used by developers who have already used the pattern to check its changes.
4. **Structural adjustments:** Introduction of field extensions and omissions to the pattern template.
5. **Problem:** A short description of the problem that this pattern solves.
6. **Motivation:** Description of the forces involved and a problematic situation intended to motivate the use of the pattern.
7. **Context:** Wide description of the environment in which the problem and solution recur and for which the solution is desirable.
8. **Applicability:** Description of the conditions wherein the pattern can be applied.
9. **Requirements:**
   - 9.1. *Functional requirements (FR)*: List of all FR organised through use cases.
   - 9.2. *Non-functional requirements (NFR)*: List of all NFR (e.g. security) organised in a SIG (Chung et al., 2000).
   - 9.3. *Dependencies and contributions*: Identification of relationships between requirements. These may be dependencies, meaning that a requirement depends on another, or contribution, meaning that a requirement contributes positively or negatively to another requirement. This is represented with a graph.
   - 9.4. *Conflict identification & guidance to resolution*: Explanation for requirements interaction and conflict resolution.
   - 9.5. *Priorities*: Definition of priorities among the requirements. This could be represented by a hierarchical structure.
   - 9.6. *Participants*: Identification and description of the actors that interact with the system.
10. **Modelling:**
    - 10.1. *Structure*:
      - 10.1.1. Class diagram: Structure of the elements of the pattern.
      - 10.1.2. Class description: Description of classes and their responsibilities.
    - 10.2. *Behaviour:*
      - 10.2.1. Collaboration or sequence diagrams: Suitable for scenarios description.
      - 10.2.2. Activity diagrams: Suitable for scenarios and overall description.
      - 10.2.3. State diagrams*:* Suitable for scenarios and overall description.
    - 10.3. *Solution Variants*: Description of alternative structural and behavioural models.
11. **Resulting context**: System configuration after the pattern application.
12. **Consequences**: Advantages and disadvantages of the pattern application.
13. **Anti-patterns traps**: Most common pitfalls that can be originated from the pattern application.
14. **Examples**: One or more application examples that illustrate the usage of the pattern: initial context.
15. **Related patterns**: List of similar patterns (describing similar problems and solutions).
16. **Design patterns**: Design or architectural patterns that can be used for further refinement.
17. **Design guidelines:** Advices on how the pattern should be implemented (without specific details).
18. **Known uses**: Known pattern occurrences and applications in existing systems. This should include at least three different systems.

Figure 1: Proposed analysis pattern template.

Each entry in the template is numbered for referencing purposes only, not representing the filling order. This order, wherein the various entries should be filled, is given by the process described in Section 3. In general, there are no systematic processes defined to help developers building analysis patterns. The pattern community is usually more interested in building patterns than in defining rules to build them. For a practitioner, however, such a process may be essential to get started, and to know exactly what steps to take to have a pattern defined in the end.

3 A MODEL FOR ANALYSIS PATTERNS SPECIFICATION

The process depicted in Figure 2 as an activity diagram, illustrates a systematic model for analysis patterns specification. This process shows what a developer should do when defining an analysis pattern. Each marked block in the activity diagram will be described next. Each activity helps filling in one entry of the template. It is not the aim of this paper to define how an analysis pattern is identified. This work presupposes that this has already been realised. The process is explained next, step by step. Due to space reasons we could not illustrate the approach, but a full example can be found in http://ctp.di.fct.unl.pt/~ja/AP-Process.pdf, where the approach is applied to the analysis pattern "*Repair of an Entity*" (Fernandez and Yuan., 2001).

Context and Problem Definition. The first activities are realized based on the pattern identification, rooted in a set of applications that were analysed beforehand. The *Name* must be generic and abstract enough, being adaptable to the same problem within several domains. The *Problem* states the reason why the pattern is being developed. A pattern only addresses one problem. If we realize that the problem can be decomposed in several self-contained sub problems, then we isolate one problem per pattern, and recursively apply this process to each identified sub-problem. The *Context* characterizes the domain in which the problem recurs, addressing its origin, main causes/reasons, and any other relevant aspect. The *Motivation* entry describes the forces that drive the pattern and gives one example that motivates the use of the pattern. *Applicability* involves enumeration of the problem core characteristics that are solved through the solution described in this pattern.

Requirements. The *Requirements* set of activities starts by identifying and describing FRs, NFRs and participants (which can be realised in parallel as they are strongly coupled). To complement the description of the FRs we can employ a use case diagram, where each use case refers to one FR or a set of FRs. Participants are mapped to actors. NFRs

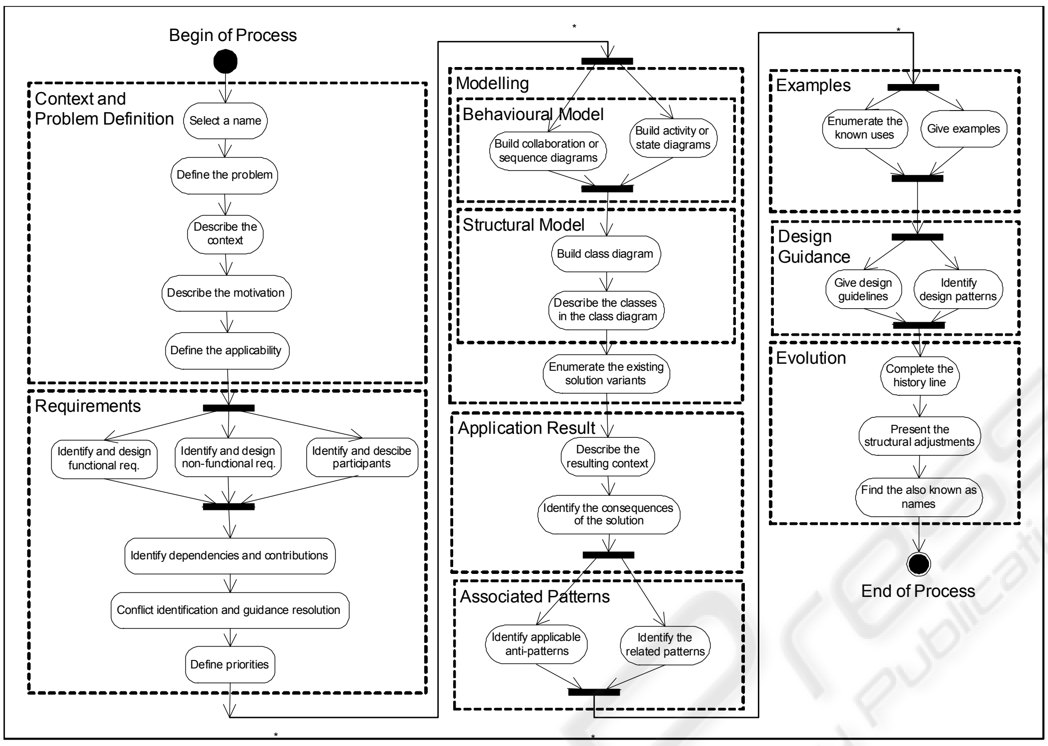
Begin of Process
Context and Problem Definition
Select a name
Define the problem
Describe the context
Describe the motivation
Define the applicability
Requirements
Identify and design functional req.
Identify and design non-functional req.
Identify and descibe participants
Identify dependencies and contributions
Conflict identification and guidance resolution
Define priorities
Modelling
Behavioural Model
Build collaboration or sequence diagrams
Build activity or state diagrams
Structural Model
Build class diagram
Describe the classes in the class diagram
Enumerate the existing solution variants
Application Result
Describe the resulting context
Identify the consequences of the solution
Associated Patterns
Identify applicable anti-patterns
Identify the related patterns
Examples
Enumerate the known uses
Give examples
Design Guidance
Give design guidelines
Identify design patterns
Evolution
Complete the history line
Present the structural adjustments
Find the also known as names
End of Process

Figure 2: The process model.

are properties that constrain the FRs, identified using catalogues as in (Chung et al., 2000). In the next step, identify dependencies and contributions, we need to find the relationships between the requirements recognized. Next, we can characterise conflicts by identifying negative contributions. Conflicts must be negotiated by assigning priorities to requirements.

Modelling. Once we have defined the requirements, their dependencies, and solved the conflicts identified, we are able to initiate the *Modelling* set of activities. This starts by specifying the behavioural model as a way to understand the dynamics of each use case and to identify the objects necessary for its "execution" with the help of sequence or collaboration diagrams. Having done that, we can then initiate the construction of the class diagram for the pattern (based on the objects identified). To *Enumerate the existing solution variants* we must provide a list of other solutions for the same problem described in the pattern. Although these solutions may look unnecessary, since they are non optimal compared with the solution presented, they should be presented since a modification on the problem or context may make them more appropriate.

Application Result. Once the modelling activity is concluded, we have enough information to describe the resulting context and the consequences of the application of the pattern. While the *Resulting Context* specifies the system configuration after the pattern application, the *Consequences* entry lists both the advantages and disadvantages of the pattern's application. The identification of the disadvantages can be used as a starting point for the application of other patterns, which would complement the current one.

Associated Patterns. Knowing the resulting context and the consequences of the pattern application we have all the information necessary to define a list of *Related Patterns* that specifies different solutions for related problems, as well as *Anti-Patterns Traps* (Brown et al., 1998). The latter helps avoid common errors in the pattern application by presenting the most common negative results.

Examples. The previous steps encourage precise definition of an analysis pattern. However, for better understanding we need concrete examples. The *Known Uses* field should enumerate at least three examples of the pattern application in implemented systems. The *Examples* field shows how the pattern was applied and all transformations necessary to the initial context so that it could be applied.

Design Guidance. *Design Guidelines* provide advice and general guidelines for the implementation step. These advices should be platform and language independent. The *Design Patterns* entry shows a list of suitable patterns that can be applied to the implementation of a pattern.

Evolution. For evolution purposes, we need to supply the requirements engineer with some extra information. The *History* entry explains all the transformations the pattern suffered, tracking the pattern's progress, since the original version. This helps developers to identify what changes have taken place. *Structure Adjustments* should include all additional extensions, all omitted fields, and the reasons for those decisions. *Also Known As* lists additional names for which a pattern is also known.

4 RELATED WORK

In (Whitenack, 1995), a pattern language is described for requirements elicitation. Guidance is provided for analysts and product developers to apply a set of techniques to produce a deeper understanding of the problem area. However, this pattern language is more appropriate to simpler pattern descriptions, not applicable to our template.

In (Robertson, 1996), an event/use case approach is used and employs a simple template for pattern description. In (Konrad and Cheng, 2002), the focus is on requirements patterns for embedded systems. In (Fowler, 1997), the concept of analysis patterns is proposed for the representation of conceptual models for commercial processes. In (Fernandez and Yuan, 2000), the Semantic Analysis Pattern presented portrays a small set of coherent use cases that describe a basic generic application. All these approaches focus on the structure of analysis patterns, and not on the definition of a process of how to build them. Our work addresses this issue by presenting a systematic process model.

5 CONCLUSIONS

This paper presented a systematic process to specify analysis patterns using a template that provides detailed information. The aim was to facilitate the developers' work in charge of the analysis patterns specifications by guiding them in this task. We believe that this approach will (a) encourage software engineers to specify patterns with better quality and (b) provide developers with more detailed information, essential to decide which pattern should be chosen. Notice, however, that it is not our intention to propose a rigid process. Adaptations are allowed if needed to follow the common practices of an organization.

As future work, we intend to adapt the process to accommodate the emerging aspect-oriented analysis specifications. With such work we envision that it shall be possible to broaden the template's applicability and usage. Furthermore, we plan to provide tool support not only for the process model presented in this paper, but also to automatically reconfigure and adjust this process to accommodate organization's particularities.

REFERENCES

Brown, W., Malveau, R., McCormick, H., Mowbray, T., 1998. *Anti-Patterns: Refactoring Software, Architectures and Projects in Crisis*, J. Wiley & Sons.

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

Gamma, E., Helm, R., Johnson, R., Vlissides, J., 1995. *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley.

Fernandez, E.B., Yuan, X., Semantic Analysis Patterns. ER'00, USA, 2000.

Fernandez, E.B., Yuan, X., An Analysis Pattern for Repair of an Entity. PLoP 2001, USA, 2001.

Fowler, M., 1997. *Analysis Patterns - Reusable Object Models*, Addison Wesley.

Konrad, S., Cheng, B., 2002. Requirements Patterns for Embebed Systems. RE'02, Essen, Germany.

Pantoquilho, M., R. Raminhos, Araújo, J., 2003. Analysis Patterns Specifications: Filling the Gaps. *Viking PLoP 2003*, Bergen, Norway.

Robertson, S., 1996. *Requirements Patterns Via Events / Use Cases*. The Atlantic Systems Guild.

Whitenack, B.G., 1995. *RAPPeL: A Requirements Analysis Pattern Language for Object Oriented Development*. Addison-Wesley.