18

Sustainable software design with design patterns

Adelino R. F. da Silva
Departamento de Engenharia Electrotécnica
Universidade Nova de Lisboa
2825 Monte de Caparica, Portugal
T: +351.1.294 83 38; Fax: +351.1.294 85 32
Email: asilva@acm.org

Abstract

Researchers in software engineering and information engineering have been struggling to modernize software life-cycle processes and to improve the quality of software. Recently, several methodologies and organizational changes have been put forward to improve systematic software specification and reuse. Design patterns offer a broader view helping us think at the architectural level. In this paper, several architectural abstractions guiding the development and reuse cycle in the production of complex software systems are introduced. We follow the design patterns methodology in the description and presentation of domain specific patterns taking the open distributed processing domain as reference. Several design patterns illustrating the mechanisms used for capturing design discipline knowledge are presented.

Keywords

Design patterns, Object-oriented methodologies, Software engineering, Distributed processing, Architecture description languages

L. M. Camarinha-Matos (ed.), *Re-engineering for Sustainable Industrial Production*

1 INTRODUCTION

Researchers in software engineering and information engineering have been struggling to modernize software life-cycle processes and to improve the quality of software. Recently, several methodologies and organizational changes have been put forward to improve systematic software specification and reuse. These methodologies are aimed at developing adaptable software components from which application software can be constructed. Object technology, for instance, has been heralded as a promising solution to software complexity. However, a growing number of researchers, managers and practitioners have begun to realize that the object-oriented approach alone does not provide the right level of abstraction. Design patterns (Gamma, 1995) offer a broader view helping us think at the architectural level. The idea of using patterns and pattern languages is borrowed from the work done in building architecture by the architect Christopher Alexander (Alexander, 1977). In Alexander's terms, "each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same twice". A pattern is then a "rule which describes what you have to do to generate the entity which it defines". Software system designers are faced with an analogous situation. Expert designers know how to reuse solutions that have worked for them in the past.

In order to simplify the process of building increasingly large and complex computing systems, international standards bodies have addressed the problem of coordinating the standardization of architectural frameworks for the integrated support of distributed applications. In particular, the Reference Model of Open Distributed Processing (RM-ODP) (ITU-T X.901, 1995) is a joint effort by the international standards bodies ISO and ITU-T to develop a coordinating framework for the standardization of open distributed processing (ODP). Open distributed processing describes systems that support heterogeneous distributed processing, both within and between organizations through the use of a common interaction model. RM-ODP is an architectural framework for the integrated support distribution, inter-working, inter-operability, and portability of distributed applications. It provides an object-oriented reference model for building open distributed systems. The general goal of ODP standardization is the development of a framework for system specification.

As a meta-standard, RM-ODP requires a supporting architecture to coordinate and guide the development of application-specific ODP standards. In this paper, several architectural abstractions guiding the development and reuse cycle in the production of complex software systems are introduced. We follow the design patterns methodology in the description and presentation of domain specific patterns, taking the open distributed processing domain as reference. Several design patterns illustrating the mechanisms used for capturing design discipline knowledge are presented. We propose the utilization of the design patterns approach as the appropriate methodology to support the development of a framework for ODP system specification. By referencing approaches that have been followed in multiple design domains, we show how design patterns abstract knowledge representation.

2 THE DESIGN PATTERNS APPROACH

Recently, software designers have discovered strong analogies between software architecture patterns, and the building architectural patterns of C. Alexander. He found recurring themes in architecture, and captured them into descriptions and instructions. He used the term 'pattern' to express the replicated similarity in a design. Patterns provide a way to share design expertise in an application-independent way, and emphasizes their potential for reuse. The pattern characterization makes room for variability and customization of the intervening elements. Alexander's pattern language (Alexander, 1977) contains over 250 patterns, organized from high level to low level. The goal in documenting patterns is to build a system of patterns. A coherent system of patterns in a design domain is often referred to as a pattern language. A pattern language in this context is not a programming language in the ordinary sense of the term, but is a structured collection of patterns that build on each other to guide and inform the designer toward well-designed architectures. A pattern language is an architectural technique because it tries to characterize relationships within and between the parts. The goal is not merely to apply a divide-and-conquer approach, but rather make a complex work understandable by organizing it in a system of well-known patterns. In a given situation there may be multiple patterns to apply. The chosen relation between the patterns is said to form a pattern language in a given problem domain. Knowledge of development design and process management has typically been kept unstructured in terms of the underlying architectural patterns for reuse. Much of this knowledge can be captured using the patterns methodology.

3 EXPRESSING DESIGN PATTERNS

Patterns are well suited to documenting design techniques, expressing architectural considerations independent of language and design methodology. A common way of expressing patterns is to use the *Alexandrian form,* which tells what is contained in a pattern. An Alexandrian-like pattern consists of the following components:

- *Name.* The pattern name is the name by which the pattern is called.
- *Problem.* This is a concise problem statement It describes briefly the problem the pattern is trying to solve.
- *Context.* It describes the situations in which the solution is more or less applicable. A pattern is expected to solve a problem in a given context.
- *Forces or tradeoffs.* In a given context, conflicting constraints and requirements may affect the solution. The pattern finds a reasonable compromise between the conflicting requirements and 'resolves the forces'.
- *Solution* The solution component describes the elements that make up the design, their relationships, responsibilities and collaborations. The solution component describes behavior as well. It documents the actions to take or structures to build.
- *Example.* Examples show how to implement the pattern to solve typical problems in given contexts.
- *Discussion.* This component describes how the pattern relates to other patterns in the language. Related patterns may provide alternatives to the proposed solution.

4 ARCHITECTURES FOR INFORMATION PROCESSING SYSTEMS

Many architectures for information processing systems, guiding complex development projects, have been proposed in the past. These architectures often emerge and are elaborated on the context of a specific development process. Knowledge gained from previous projects is typically kept in one of the forms: human expertise and project-oriented documentation. The goal of abstracting from the peculiarities of particular designs is rarely considered. What is new in the patterns approach is its commitment to produce a 'handbook of useful solutions'. Collecting a patterns-catalogue systematically becomes part of the quality program of a development team or organization. We present below three design patterns for the specification of software architectures.

4.1 Pattern 1: abstraction levels

- *Problem*: How to specify the abstraction levels relevant to carry out a design discipline ?
- *Context*: A design and implementation discipline needs to be outlined for a well-understood domain. The analysis phase of the system development process is often called conceptual modeling, since it results in a conceptual model of the system to be implemented.
- *Forces*: A design discipline may have several levels of abstraction. Different design methodologies are used to manage design problems at different levels of abstraction. Although design is simpler at the higher abstraction levels, difficulties arise because it is necessary to manage object interactions between levels. Design at higher levels of abstraction becomes dependent on performance characteristics only observable at lower levels of abstraction.
- *Solution*: For each design object in a design domain name and characterize the target abstraction levels more suitable to carry out the design objectives. List the names of the abstraction levels by order of abstraction
- *Example*: RM-ODP defines a division of ODP system specification into five abstractions or viewpoints, in order to simplify the description of complex systems. Each ODP viewpoint provides a representation of the system with emphasis on a specific set of concerns (see Figure 1). The five ODP viewpoints are: the enterprise viewpoint (EntV), the information viewpoint (InfV), the computational viewpoint (ComV), the engineering viewpoint (EngV), and the technology viewpoint (TecV). EntV focuses on the purpose, scope and policies of the system. InfV focuses on the semantics of the information and information processing performed by the system. ComV is concerned with the description of the system as a set of objects that interact at interfaces, enabling distribution through functional decomposition. EngV focuses on the mechanisms and functions required to support distributed interaction between objects in the system. TecV focuses on the choice of technology in the system.
- *Discussion*: This pattern is common in multiple design domains. In (Kleinfeltd, 1994), for instance, the pattern has been applied to electronic design (CAD) framework systems. Fishwick (Fishwick, 1995), has described a related pattern in the context of simulation model design, under the name 'multimodeling'. He defines multimodeling in the following terms: "Multimodeling is a modeling process in which we model a system at multiple levels of abstraction. ... With abstraction we are concerned with many levels - each of which has the potential to be independently simulated". Kerth (Coplien, 1995), uses the Caterpillar's

Fate pattern to guide the transformation of a system from the analysis stage into the design stage. The pattern suggests a methodology consisting of an analysis phase and a design phase.

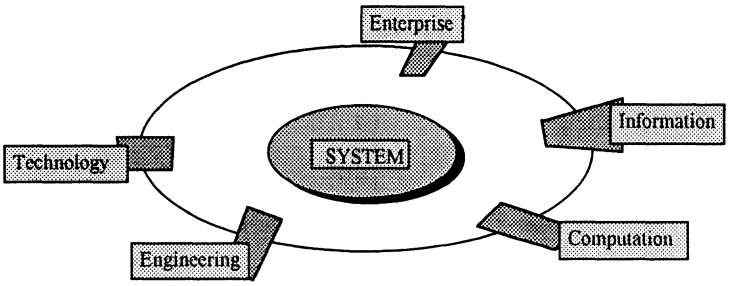

Figure 1 ODP viewpoints.

4.2 Pattern 2: design domains architecture

- *Problem*: How to create a design architecture in terms of design domains and their specializations, so that the system can meet its objectives and the requirements satisfied.
- *Context*: In this pattern we are concerned with many levels of abstraction, each of which has the potential to be independently designed and implemented. It is well known that small projects are easier to manage than big projects. A layered architecture is beneficial when many design objects are involved. A design domain is a set of design objects. These objects exist at different levels of abstraction. The design domains in a given design discipline can be arranged in a design domain hierarchy. When moving down a hierarchy, one refines the design domain. When moving up, one abstracts the design domain.
- *Forces*: The question of how to create a layered architecture emerges in this situation. A good architecture simplifies the implementation of the system. Some design methodologies are more appropriate to characterize structure than others. A design domain may be represented by different abstractions at the same level, corresponding to different perspectives of the system. In this case, a mapping relationship may be required to integrate both perspectives in a coherent system.
- *Solution*: Arrange the design domains for a given design discipline in a design domain hierarchy. The top level domain corresponds to the design discipline itself. Lower level design domains correspond to specializations of the parent level design domain. This specification must be done at the appropriate target abstraction level for the specific domain. The objective is to separate the concepts and notation used for different scopes, or levels of granularity. Follow the object-oriented approach, in order to identify key roles and interactions between levels.
- *Example*: Two main structuring approaches are used in the ODP architecture: viewpoints and distributed transparencies. These two approaches provide orthogonal representations. A viewpoint is a subdivision of the specification of a complete system relevant to some particular area of concern. On the other hand, distributed transparencies provide standard

mechanisms to access heterogeneous systems and components. The standard mechanism is 33said to provide a transparency. With transparencies the application designer works in a world of independent concerns. For instance, access transparency masks differences in data representation and invocation mechanisms to enable interworking between objects. Eight ODP transparencies have been defined (see (ITU-T X.901, 1995) for details).

- *Discussion*: The five ODP viewpoints are not organized in a hierarchy or in layers but represent the complete set of related viewpoints of a system architecture. However, they effectively provide different levels of abstraction. Rettig et al. (Rettig, 1993), in the context of the PADRE project planning and development process, uses a similar pattern based on modules: "Planning proceeds top-down until the work has been divided into many small pieces called 'modules'. Each team member uses the module-level process definition (...) to manage the progress of his or her current module, while the manager guides the progress of the whole series of modules toward the final goal". The 'Pedestal' pattern (Coplien, 1995), addresses the problem of how to create a layered architecture for a mechanical-process control system. He considers that domains should the selected and ordered based on the domain's purpose in the system.

4.3 Pattern 3: design problem decomposition

- *Problem*: How to decompose complex design objects to meet the domain requirements specifications.
- *Context*: A design discipline encompasses complex design objects of some broader type. A given domain decomposition works within a given abstraction level, which is adequate for carrying out the design objectives. The interactions between sub-domains are well-understood.
- *Forces*: It is possible that a design domain may have several possible decompositions Design domains may decompose into others of the same type or domain. The iterative nature of the design process may require a refinement of the modular decomposition, both in terms of abstract levels and in terms of domain decomposition.
- *Solution*: Choose the design strategy most appropriate to decompose complex design objects to meet the domain requirements specifications. Additional considerations, such as risk analysis, time to market, facility of reuse and easy maintenance, may strongly impact this choice as well. Several methodologies may be followed to partition a domain, namely object-oriented, object-based, structured, data flow diagrams, control flow diagrams, etc. Traditional process-oriented models stress sequencing of activities to accommodate chronological dependencies. For simple decomposition problems, a strict functional decomposition may be sufficient. In this case, apply design objective decomposition to break up the design goal into sub-goals which apply to the same whole design object. In general, however, more powerful methodologies are required. Object-oriented design emphasizes the discipline of identifying key roles and interactions in a design domain. Once the principal roles have been identified, responsibilities are assigned to each of these roles, and the circumstances under which roles interact with one another are defined.
- *Example*: In order to cope with the complexity of large ODP systems, it is necessary to provide a common framework for partitioning overall management. Management domains are used in RM-ODP to specify boundaries of management responsibility and authority.

They permit a set of managed objects to be controlled under a common policy. Each management domain specifies the management policy for a group of managed objects. On the other hand, for each abstraction it is necessary to define a structured set of concepts in terms of which that representation (or specification) can be expressed. This set of concepts provides a language for writing specifications of systems from that abstraction point of view. The RM-ODP enterprise language, for instance, introduces basic concepts necessary to represent an ODP system in the context of the enterprise in which it operates. In order to express the objectives and the policy constraints, the system may be represented by one or more enterprise objects, and by the roles in which these objects are involved. These roles, represent, for instance, the owners, developers, customers, and users involved in the enterprise activities. Moreover, in order to link the performers of the various roles and to express their mutual obligations, the enterprise language introduces the notion of contract. For example, the enterprise language for a distributed client-server system with one operator and two users might be specified in the following terms (see Figure 2): (i) Two management domains; (ii) Roles of the participating agents in terms of the system objectives (human-operator role, system-provider role, system-accessor role, and human-user role); (iii) Interactions between agents (represented by links in Figure 2); (iv) Contracts representing the policies and requirements governing the interactions between agents. Similar decompositions may be specified for the each RM-ODP viewpoint, in terms of its specific language.

- *Discussion*: Rettig et al. (Rettig, 1993), after breaking the planning and deveplopment process into modules, applies a *micromanagement* technique to each module based on five sequential milestones: plan, approve, do, review and revise, and evaluate. Fishwick (Fishwick ,1995), decomposes the process of making tea in three sequential stages: heating water, inserting bag in water, and steeping tea. This process is the refinement of a top-level abstraction concerned with the 'Making tea' objective. This pattern has similarities with the 'Divide and Conquer' pattern proposed by Coplien for organizational patterns (Coplien, 1995). The patterns 'Decouple Stages' and 'Hub, Spoke, and Rim' (Coplien, 1995), provide indications on how to decompose a domain. Several methodologies have been proposed to address the question of how to design behavioral composition in object-oriented systems (Rumbaugh, 1991). They emphasize the elaboration of a semantic net of domain objects in terms of interacting entities.

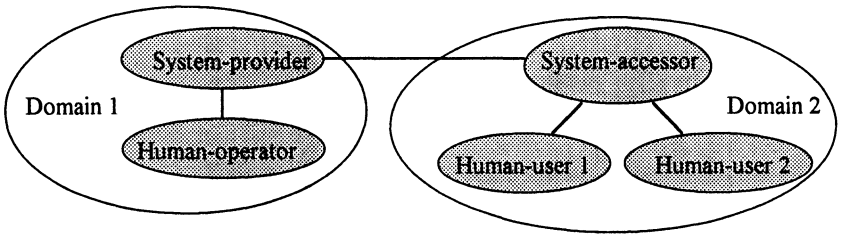

Figure 2 Enterprise viewpoint for a client-server system.

5 CONCLUSION

Design patterns are an emerging methodology for guiding and documenting system design. A large range of patterns for software architecture, design and implementation has been produced by the pattern community. Patterns covering the activities of design process management and project development are beginning to emerge as well. As the repository of such patterns grows, it will be possible to produce a pattern system for the organization and categorization of related patterns. By relying on architectural abstractions to capture existing design knowledge, patterns are a promising approach to guide the development and reuse cycle in the production of complex systems.

6 REFERENCES

Alexander, C., Ishikawa, S., and Silverstein, M. (1977) *Pattern Language - Towns-Buildings-Construction*. Oxford University Press.

Coplien, J. and Schmidt, D. (1995) *Pattern Languages of Program Design*, Addison-Wesley.

ITU-T X.901 (1995) *ODP Reference Model Part 1: Overview*, ISO/IEC 10746-1.

Fishwick, P. (1995) *Simulation Model Design and Execution - Building Digital Worlds*, Prentice-Hall.

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

Kleinfeltd, S., Guiney, M., Miller, J.K. and Barnes, M. (1994) Design Methodology Management, *Proc. IEEE*, vol. 82, n° 2, pp. 231-250.

Rettig, M. and Simons, G. (1993) A project Planning and Development Process for Small Teams, *Comm ACM*, vol 36, n° 10, pp. 45-55.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. and Lorensen, W. (1991) *Object-Oriented Modeling and Design*, Englewood Cliffs, Prentice Hall.

7 BIOGRAPHY

Adelino R. F. da Silva is an assistant professor in the Electrical Engineering Department at Universidade Nova de Lisboa, Lisbon. He received his PhD in Systems Engineering in 1989 from the Universidade Nova de Lisboa. Previously, he had been a Fulbright teaching and research assistant with the University of Pennsylvania. His main interests are Object-oriented methodologies, User-interfaces, Interactive systems and Human-computer interaction. Dr. Adelino Silva is a member of ACM and IEEE-CS.