An object-oriented pattern language for engineering design

Y.P. Khanal and R.O. Buchal
Department of Mechanical and Materials Engineering
The University of Western Ontario
rbuchal@eng.uwo.ca

Abstract

Psychological inertia, biases and incomplete knowledge can lead engineering designers to choose a sub-optima design. The use of design patterns can help engineering designers find better solutions.

Design patterns are recurring/reusable design solutions that are known to work in a particular design situation or a context. Patterns represent both recurring problems and recurring solutions together with their relationships. A design pattern language is a collection of related design patterns covering a particular design domain. This paper describes an object-oriented framework and methodology for the construction and use of pattern languages for the design of technical systems.

Key words: Engineering design patterns; Engineering design pattern language; Engineering design; object-orientation.

1 Introduction

A technical system is a man-made artefact that performs functions to achieve desired goals [1]. The primary concern for designers in today’s competitive world is to design technical systems effectively and reliably so that the quality of design is not compromised by various factors such as time-to-market pressure, risk undertaking and so on. In achieving this goal, tools are needed to help designers efficiently synthesize good design concepts that are historically proven to be reliable and valid.

According to axiom of object representation [2], everything is represented as an object, including a technical system, its functions, a design problem, a design solution and so on. Patterns are recurrent (reusable) objects and a pattern language is a network of patterns [3]. How we define the recurrent objects and create a network matters in the design of a technical system.

General engineering design did not have the concept of pattern language until Salustri proposed it in 2005 [4]. In this paper, we propose descriptive and prescriptive approaches to using patterns and their associated pattern languages in design.

Descriptive relates to “what something is”. In the descriptive approach, each pattern can be a design description at a certain abstraction level. The pattern embodies in it the problem, solution and context [3]. The pattern sets the protocol for high level collaboration between other objects to form a solution. The solution captures only the invariant properties or scenarios of the solution description [3], and the designer may come up with different manifestations of the solution. One or more patterns set contexts for one or more other patterns in this approach.

Prescriptive means “how something should/can be done”. In the prescriptive approach, the recurrent objects may form a network with logical relations. The object relations are explored for analogical reasoning in design [5].

A pattern language can be created using both approaches and has great significance in engineering design. It helps designers to remove design biases, come up with solutions faster, import/export cross domain knowledge and so on. A formal framework is required to create and manage the pattern language. We propose a framework for creating an object-oriented design pattern language covering a particular domain that can exploit both object-oriented features and object relational features.

2 Descriptive approach to object-oriented patterns

Alexander et al. [3] introduced the concept of design patterns, and proposed a pattern language for design of towns and buildings. Each pattern in the language is linked to other patterns that are larger patterns and
smaller patterns with respect to the pattern. Object-oriented design patterns [6] have been widely used in software engineering with new patterns being continuously developed and applied in the design of software. A descriptive software pattern is like a standard fastener or machine element that can be used without needing to be designed from scratch. Object-orientation has promised benefits of reducing complexities. Due to the benefits, it is useful to formalize and standardize the object-oriented patterns for the general technical system design.

A descriptive pattern is an object that embeds in it the design context, problem and solution. A descriptive pattern represents a design solution at some level of abstraction. It is equivalent to referring to a design catalogue and selecting a particular solution to a problem in a design context. For example, if the design problem is ‘how to provide power?’ and the solution corresponds to the selection of a ‘motor’, the problem-solution pair in the design context represents the descriptive pattern. All object-oriented software patterns are descriptive, as are the reusable catalogues of design for general engineering design.

The power of descriptive patterns lies in the abstraction of objects. The abstraction creates a new class of objects based on their predicable properties. The designer must be able to trigger design concepts as objects at a level of abstraction suitable for design. Some abstractions may lead to a very high level definition of a design problem and/or solution and need to be further refined. For example, a technical system can be represented as a black box that transforms mass, energy and information (MEI). Such a high level of abstraction is rarely useful but it directs the designer in further refining the problem or solution.

According to Hubka and Eder [7], a technical system produces and consumes the operands in relation to its operating environment (and other influencing operators). This can be represented by a producer-consumer pattern. The producer and the consumer are the generic objects of a pattern at the highest abstraction level.

A technical system as a producer or consumer can be viewed as an object surrounded by an environment [2]. The producer-consumer pattern may apply to objects in the technical system, and objects in the environment. Within the technical system, there may be multiple interacting objects. If the pattern application crosses the boundary, object(s) of the technical system may interact with objects of the environment.

The producer-consumer pattern is also known as action-reaction (Zeng 2002), input-output, and source-sink and so on.

Figure 1 shows an example of an object-oriented producer-consumer pattern. The presentation form of this pattern is as follows:

**Context:** Design synthesis of technical system objects
**Problem:** How to model a technical system in which the operands in the form of MEI are produced and consumed?
**Solution:** The producer produces MEI and the consumer consumes MEI.

![Figure 1. A producer-consumer pattern.](image)

The automotive serpentine belt shown in Figure 2 is an example of a producer-consumer pattern. The crank produces power that is distributed to various consumers: the three idlers IDL, alternator ALT, water pump & fan, and the environment. The environment is the abstract form of consumer. Anything not explicitly represented by the technical system objects in Figure 2 can be included in the environment. The distribution must follow the energy conservation principle. The following expression holds true for the case of serpentine belt system.

\[ P_{\text{crank}} = P_{\text{idl}_1} + P_{\text{idl}_2} + P_{\text{idl}_3} + P_{\text{waterpumpfan}} + P_{\text{alt}} + P_{\text{env}} \]
where

$P_{crank}$: Power produced by the crank shaft

$P_{al}, P_{el}, P_{amb}, P_{waterpump}, P_{fan}$: Power consumed by respective design objects

$P_{env}$: Power consumed by the environment

3 Prescriptive approach to object-oriented patterns

Prescriptive patterns support analogical design. Analogical design involves recalling and transfer of knowledge from one design context to another, where the transfer can occur in the service of any design task in the new context [8]. It means that the designers look at existing examples to find solutions. In the object-oriented world we discuss everything in terms of objects and their relations, so the basis of such analogy is to compare the object relations.

A prescriptive pattern is related to function-modeling, because the recurrent objects (the problems or solutions) can be expressed as functions [9] as shown in Figure 3. Both problem and solution are classified according to the object-oriented inheritance relation.

\[ \text{Function} \]

\[ \text{Problem} \rightarrow \text{Solution} \rightarrow \text{Embody} \rightarrow \text{Inventive Principle} \]

\[ \text{Purpose} \rightarrow \text{TS Function} \]

Figure 3. Functional abstraction of problem and solution.

Function modeling only captures the functional interrelations of the technical system, without considering its physical structure. Usually, in the early function-modeling, functions (kinds of problems) do not associate with their solutions or solution patterns, and only the functional interrelations are established. Bytheway [9] proposed a functional modeling technique based on the use of intuitive logical relations ‘how-why’, ‘how-else’ and ‘when-if’, I-TRIZ functional modeling [10] defines only two types of function: useful and harmful. The functions are related by two types of relationship: produces and counteracts. In many technical systems, functions exist for the exclusive purpose of counteracting harmful functions. Contractions are identified by functions that produce both useful and harmful functions.

A recurrent solution is a solution pattern that can be represented by a hierarchy of objects. For example, the object ‘energy transformer’ as shown in Figure 4 can be hierarchically decomposed with inheritance to have several variations or specializations as shown in Figure 5.

![Figure 4. Transformation of various types of energy through an energy transformer.](image)

![Figure 5. Transformation of various types of energy through an energy transformer.](image)

3.1 Structure and organization of objects of a prescriptive pattern framework
Design involves the process of mapping from problem to solution and vice versa. A design problem or problem in short can have several connotations. It may refer to the ‘purpose’ of design, and also may refer to the ‘function’ of a technical system (Figure 3). The function of a technical system implies ‘what the technical system and/or its subsystem does’.

A solution implies the solution to a problem. It can also have several connotations, such as an inventive principle, an embodiment and so on (Figure 3). In Figure 6 the relationship between the problem and solution are shown by the unidirectional arrows running from problem to solution and vice versa. The stereotype <<How>> relates problem to solution (How is the problem solved?), the arrow suggesting the reading direction. The stereotype <<Why >> relates solution to problem (Why is the solution required?). This distinction is useful since many design innovations result from having a new solution, such as a new material or process, and identifying potential problems or applications.

Figure 6. Collaboration between problem, solution and illustration objects.

Figure 7 shows a function network, with functions related using Bytheway’s intuitive logics. We supplement these with a new relation, ‘why else’. Figure 8 shows how intuitive logics can be combined with I-TRIZ functional modeling (the intuitive logical relations are shown within the braces). It is noted that the two approaches to create the function network are complementary.

Figure 7. Function network with intuitive logical operators.

Figure 8. Function network with ITRIZ relational operators (operators within the braces are intuitive logics).

Problem and solution are the most commonly used lower level abstraction of functional terms. Both problem and solution can become patterns known as problem pattern and solution pattern respectively if they are recurrently used in design contexts. We create a domain for the set of the problem patterns known as the class ‘pattern’ in short. An instance of pattern and an instance of problem can be the same object. Similarly, a domain for the recurrently used solutions (solution pattern) could also have been created as a class. However, doing so does not have much significance beyond creating an extra class, as recurrence can be effectively maintained through unidirectional mapping from problem (or pattern) to solution. The mapping from problem to solution is generally agreed as the first step in design. No separate mapping from recurrent solution to problem is required.

3.2 Relational objects of the pattern framework

Four fundamental objects have been used from their respective hierarchies in Figure 6. These are goal,
pattern, embodiment and illustration. Since the four fundamental objects are related in a number of ways, such as recursive relation and multiple binary relations, other relational objects are created to represent the relations. The relational objects (labelled by blobs) appear in between the fundamental objects for m-n binary relations as shown in Figure 9. The objects are also related by intuitive logics. If the relation is recursive, its name appears as “xxxDecomposition”. Decomposition of goal and pattern objects is exclusively driven by the intuitive logics. Contains is an aggregation relation, i.e., an embodiment consist of other embodiments.

![Figure 9. Fundamental objects of the pattern framework.](image)

A prototype system has been developed to demonstrate function modeling through various object relations. The contents and illustrations have been taken from the example case study of containment ring design with Innovation WorkBench® (IWB) by Ideation International Inc.[11]. IWB implements the TRIZ methodology. We have examined the approaches of organizing functional relations by the IWB with respect to the logical relations proposed in this paper. However, in the IWB, there is no formal documentation about the organization of the functional relations. Further, the approaches we apply support object-orientation that enables designers explore functional relations with respect to object-oriented features such as inheritance and polymorphism. The screen capture of the prototype system is shown in Figure 10.

![Figure 10. A screen capture of pattern application supporting design concept generation.](image)

### 4 Summary and conclusion

A pattern is a recurrent documented best practice represented as an object which has been used over and over in the history of design. The pattern framework helps designers in the following points:

1. to be confident in attacking a design problem at hand
2. to solve the problem faster
3. to support or reduce the effect of design biases
4. to allow designer to explore more alternatives and make better design decisions.

This paper proposes creating design patterns and languages with two approaches: descriptive and prescriptive, using object-oriented abstractions. The framework supporting the application of patterns in a prescriptive way has been implemented as a prototype computer tool.

Relational data schema has been presented to create objects in the framework. The relational data are used to create respective objects. New patterns can be included in the framework by designers themselves in future without involving computer programmers. It is believed that the extension mechanism is supportive of pattern mining.

### References