A Pattern Oriented Technique for Software Design

D Janaki Ram, K N Anantha Raman, K N Guruprasad
Indian Institute of Technology
Madras
e-mail: djram,araman,guru@lotus.iitm.ernet.in

Abstract

Design patterns can be considered as elements of complex software systems. Crafting these software systems using design patterns calls for a suitable design methodology or technique. Existing design methodologies do not serve this purpose well. This paper proposes a technique which helps in developing software systems usign design patterns.

Introduction

Design patterns [3] constitute an important technique for capturing existing software designs for effective reuse. A design pattern describes a group of collaborating objects and classes which are customized for solving specific design problems. At present, these patterns are available as a catalog, in which an informal description of their usage is provided. However, an effective usage of design patterns during the design process is left to the intuition and experience of the designer. There is a need for a technique which helps in systematic use of patterns, during the design process [2].

This paper proposes a technique for software design, which makes use of patterns in a systematic manner. This technique is called Pattern Oriented Technique (POT). It helps in arriving at a set of design solutions for a given design problem. one of these solutions can be chosen.

Features of POT

The key features of POT are explained below.

- POT is pattern oriented.

  Most of the existing methodologies consider classes and objects as the building blocks of an object oriented design. Some of these methodologies group a set of classes as class categories or subsystems [1][8]. However, these subsystems or class categories are not treated as reusable units. They are used only as a mechanism for managing structural complexity. In contrast with these, POT holds patterns as the building blocks of an object oriented design. It identifies patterns in a set of colloborating objects or classes by making use of available design knowledge.
- POT makes use of design handbook.

  In many engineering disciplines which have reached a stage of maturity, designers do not always solve a problem from scratch [5]. A large body of knowledge is available as a handbook which a designer can readily consult. These handbooks help a designer in arriving at good designs making use of past experience. A case in point is Perry's handook on chemical engineering [6]. An outline of a procedure to construct a design handbook based on patterns was suggested recently [4]. POT makes use of this handbook for identification and selection of patterns.
- POT supplies a set of solutions.

  Unlike many of the existing design methodologies which provide a single solution for a design problem, POT provides a designer with a set of solutions for a given design problem. It also indicates the tradeoff among the solutions in terms of a few important characteristics.
- POT supports framework design and documentation.

  By selectively applying the steps of POT, one can arrive at a framework design. The abstract classes in this framework can be subclassed to obtain a concrete design. Since such a framework can be seen as an organized collection of patterns, it is easy to document it in terms of patterns. This is an important step towards integration of design patterns and frameworks [7].

Steps of POT

The steps of POT are described and illustrated with an example of designing a printer subsystem, whose requirements specification is as follows:

Design a system for printing files. The files are sent to the printer server by the user. The printer server keeps track of all the available printers and gets the file printed through one of them.

Identify classes

Identify the classes by selecting suitable nouns from the requirements specification. In the example, the following classes are obtained.

- file (F)
- printer server (PS)
- user (U)
- printer (P)

Identify responsiblities

Consider every class as a server which is responsible for providing certain services. Identify these responsiblities. In the example, the following responsibilities are identified.

- file
  - know about its type, size etc.
  - provide a copy of itself when requested.
- printer server
  - maintain information about the set of printers available.
  - accept files from user and send them to a suitable printer for printing.

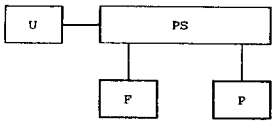

Figure 1: Class Interaction Diagram

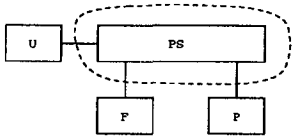

Figure 2: Singleton Pattern

- user
  - decide the set of files to be printed.
  - issue commands to the printer server.
- printer
  - know about its status, capablity etc.
  - print a file.

Identify interacting classes

Based on the responsibilities of the classes, identify all the pairs of classes whose objects interact between themselves. In the example, the following pairs are identified.

- user, printer server
- printer server, file
- printer server, printer

The classes and their interactions are shown in Figure 1. The classes are represented by rectangles and the interacting classes are connected with lines.

Identify class groups

A class group is a collection of classes, such that any class in a collection interacts with at least one class in the remainder of the collection, if the remainder is non empty. The size of a class group is the number of classes in it. Identify the class groups. In the example, the following groups are identified.

- groups of size 1
  - file
  - printer server
  - user
  - printer
- groups of size 2
  - user, printer server
  - printer server, file
  - printer server, printer
- groups of size 3
  - user, printer server, file
  - user, printer server, printer
  - file, printer server, printer
- group of size 4
  - user, printer server, file, printer

Identify class group interaction

For every class group, identify the interactions within the class group as well as the characteristics of the class group. Describe these in a couple of sentences. In the example, following interactions are identified.

- printer server
  - only one printer server exists in the system.
- user, printer server
  - the user issues print requests to the printer server.
- printer server, file
  - the printer server requests the file to provide a copy of itself.
- printer server, printer
  - the printer server maitains information about the set of printers available. It selects one of them whenever a file is to be printed.
- user, printer server, file
  - the user issues print request to the printer server, providing it with a file.
- user, printer server, printer
  - the user request the printer server to print a file. The request is forwarded to one of the printers by the printer server. The user is not aware of which printer fulfils the request.
- file, printer server, printer
  - the printer server obtains a copy of the file and sends it to one of the available printers.
- user, printer server, file, printer
  - the user issues print requests to the printer server, providing it with a file. The printer server keeps track of the printers available, selects one of them and sends a copy of the file to it.

Specify the class group interaction at an abstract level

Recast the sentences describing the class group interaction as well as the characteristics at an abstract level. In the example, following abstract descriptions are derived.

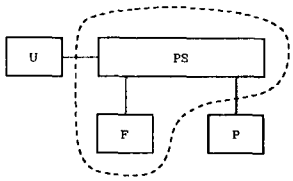

Figure 3: Prototype Pattern

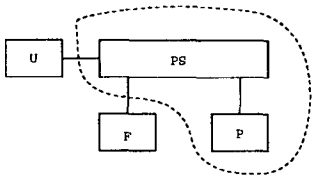

Figure 4: Iterator Pattern

- printer server
  - only one object of the given type exists in the system.
- user, printer server
  - an object issues a command to another object. The object accepting the command is known.
- printer server, file
  - an object requests a copy of another object.
- printer server, printer
  - an object maintains a list of a set of objects of a given type. It iterates through the list of objects when required.
- user, printer server, file
  - an object issues command to another object, with neccessary information.
- user, printer server, printer
  - an object issues command to another object. The object accepting the command is known. The object fulfilling the command is not known.
- file, printer server, printer
  - an object gets a copy of another object, sends it to one object in a list of objects.
- user, printer server, file, printer
  - an object issues a command to another object, with neccessary information. The object receiving the command gets the command fulfilled with the help of other objects.

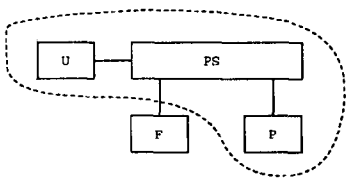

Figure 5: Chain of Responsibility Pattern

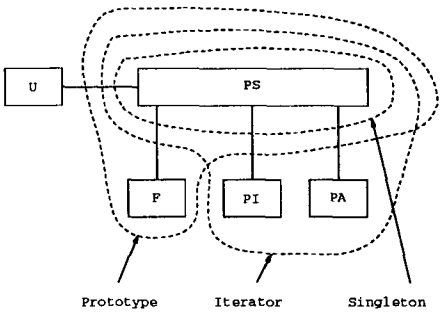

Figure 6: Rough Design I

Identify design patterns

Compare the abstract descriptions of the class group interactions with the intent of the existing design patterns. For every description, identify all the patterns which match. In the example, the following patterns are identified. These patterns are shown in Figure 2,3,4,5.

- printer server : singleton
- printer server, file : prototype
- printer server, printer : iterator
- user, printer server, printer : chain of responsiblity

Obtain rough designs

From the class interaction diagram, obtain rough designs by making use of the patterns identified in the previous step. For every pattern identified, introduce new classes or remove existing classes in the class interaction diagram, based on the pattern structure. If there exists two or more patterns which affect the same set of classes such that the patterns cannot co-exist, then each of these patterns helps us derive a new rough design.

In the example, we get two rough designs. These are shown in Figure 6 and 7. In Figure 6, PI and PA stand for printer interator and printer aggregate respectively. In rough design I, there exists a separate printer server which forwards every print request to a suitable printer. In rough design II, we have a chain of printers. All the print requests are submitted to the first printer in the chain. The request is either accepted by the printer or forwarded to next printer in the chain. The last printer is forced to accept all the print requests received.

Calculate the tradeoff

For every rough design(RD), a quantitative measure of four

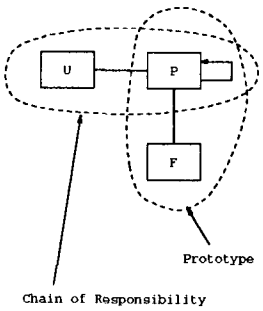

Figure 7: Rough Design II

characteristics namely coding effort(CE), static adaptability(SA), dynamic adaptability (DA) and performance(P) are derived. This is done by adding the values of these characteristics for the patterns present in the rough design.

Coding effort is a measure of the amount of coding that must be done for implementing a pattern in a programming language. Static adaptability is a measure of easiness with which a pattern can be adapted to a particular context at the time of coding. Dynamic adaptability reflects the ease with which the behavior of a pattern can be modified or adapted at runtime. Performance is a measure of the speed with which a pattern delivers services expected of it. The basis for considering these characteristics and the procedure for calculating them is given elsewhere [4].

In the printer subsystem example, rough design I contains Prototype, Iterator and Singleton pattern. Rough design II contains Prototype and Chain of Responsibility pattern. This is shown in Table I. By using the design handbook [4], the quantitative values of the characteristics for these patterns are obtained, as given in Table II. Table III is derived from Tables I and II and shows the values of the characteristics at design level. One of the designs is choosen based on these values.

Table I

RD	Pattern
I	Prototype Iterator Singleton
II	Prototype Chain of Responsibility

Table II

Pattern	CE	SA	DA	P
Prototype	140	150	100	-150
Iterator	200	200	80	20
Singleton	110	-200	0	200
Chain of Responsibility	70	100	100	0

Table III

RD	CE	SA	DA	P
I	450	150	180	70
II	210	250	200	-150

Obtain detailed design

Identify the attributes and the operations for the classes in the rough design. Subclass the abstract classes of the patterns. Identify the relationships between the classes. Refine the rough design accordingly.

Conclusion

A new technique for software design is proposed and described. An example is provided to illustrate the steps of this technique.

While calculating the tradeoff between rough designs, numerical values resulting from a simple addition of the characteristics of the patterns may not reflect the characteristics of the rough design accurately. This may be due to influence of classes which belong to more than one pattern, as well as the classes outside every pattern.

References

[1] Booch, G., *Object Oriented Analysis And Design With Applications*, Benjamin/Cummings Publishing Company Inc, 2nd edition, 1994, pp. 181-184

[2] Fraser, S., Booch, G., Buschmann, F., Coplien, J., Kerth, N., Jacobson, I., Rosson, M.B., "Patterns : Cult to Culture ?," in *Proceedings of OOPSLA'95*, Austin, Texas, USA, 1995, pp. 231

[3] Gamma, E., Helm, R., Johnson, R., Vlissides, J., *Design Patterns - Elements of Reusable Object-Oriented Software*, Addison-Wesley Publishing Company, 1995

[4] Janaki Ram, D., Anantha Raman, K. N., Guruprasad, K. N., Suchitra Raman, "A Methodology for Constructing a Design Handbook for Object-Oriented systems," in *Proceedings of the Third Annual Conference on the Pattern Languages of Programs*, Monticello, Illinois, USA, Sep 1996, Volume 9, Chapter 5, Section 9.5.1

[5] Marciniak, J. J., *Encyclopedia of Software Engineering - Volume 2*, Wiley-InterScience Publication, 1994, pp. 930-940

[6] Perry, Chilton Editors, *Chemical Engineer's Handbook*, 5th Edition, 1973

[7] Schmidt, D. C., Johnson, R. E., Fayad, M., "Software Patterns," *Communications of the ACM*, Oct 1996

[8] Wirfs-Brock, R., Wilkerson, B., Wiener, L., *Designing Object-Oriented Software*, Prentice-Hall of India Private Ltd., 1996, pp. 131-160