PERIODICA POLYTECHNICA SER. EL. ENG. VOL. 47, NO. 3–4, PP. 205–212 (2003)

DESIGN PATTERN MATCHING

Dániel PETRI and György CSERTÁN
Department of Measurement and Information Systems
Budapest University of Technology and Economics
H–1521 Budapest, Hungary
e-mail: dpetri@mit.bme.hu,csertan@mit.bme.hu

Received: Oct. 1. 2003

Abstract

Since the beginning of software development re-usability has been an important aspect. Applying reusable elements the software does not have to be developed from scratch, proved solutions can shorten the software development life cycle and make applications safer. There are several forms of re-usability like function libraries, class libraries, design patterns, component technologies and enterprise templates, among which this article deals with design patterns. Our intention is to help developers to find the appropriate design patterns without having to study the tremendous amount of existing patterns. Therefore we complete the design pattern metamodel with some additional information about the pattern's purpose and responsibilities. This completion allows simplified search of patterns and query of design pattern properties.

Keywords: design patterns, formalization, query, search.

1. Design Patterns

Design patterns in general capture low-level object-oriented concepts: the distilled experience of expert designers. They document proved designs without involving domain-specific features, thus specifying only essential aspects. The literature that documents important design patterns is tremendous and rapidly growing. [5] is a pioneer work in this field.

Design patterns have three main parts:

- A problem within a given context,
- A solution skeleton to this problem, and
- The consequences of the solution listing benefits and drawbacks.

These three main sections are divided into mandatory and optional subsections. The header of a pattern may list some keywords, the categorization of the pattern and other related patterns. The problem section also lists the intent and the motivation. The solution section contains the structure of the solution, the participants, their responsibilities and their collaborations. Object-oriented design patterns may also contain implementation guidelines and source code fragments.

The description of design patterns, as they originate from the architecture [2, 1], has many informal or narrative sections, and contains UML[1] Class and

[1]Unified Modeling Language: the de facto standard visual object-oriented modeling language

Sequence diagrams that describe the static and dynamic behavior of the solution. Usually, the problem and context description, the solution and the consequences sections are described in a natural language, usually in English. Examples of design patterns can be found e.g. in [5, 3, 11].

Patterns related to the same problem domain can be arranged in so-called *pattern languages*. A pattern language is not a formal language, but rather a collection of interrelated patterns, though it does provide a vocabulary for talking about a problem domain. A pattern language may define the order in which patterns should be applied to a particular problem. It also defines pattern relationships, and gives instructions about which patterns should be used together and which patterns exclude the use of other patterns.

2. Tools Concerning Design Patterns

There are many tools and tool ideas with the purpose of:

- Automating the process of detecting the appropriate design patterns,
- Checking whether the implementation conforms to the description of a pattern,
- Identifying existing and new design patterns in an implementation,
- Determining pattern relationships, e.g. a pattern is a specialization of another pattern (meaning that one pattern is a special case of another).

Our idea belongs to the first group, i.e., it simplifies the process of design pattern detection based on a system specification or description.

3. Formalization Efforts

For these tools the original descriptive form of design patterns is not always appropriate. There are many formalization efforts around design patterns.

In [4] LePUS, Language for Patterns Uniform Specification is introduced. The main purpose of this language is to describe the generic solution indicated by a design pattern, which involves a set of participants and their collaborations.

[7] presents a technique for formalizing design patterns using a method based on attribute grammars. This technique allows design pattern implementations to be identified in the source code, and supports automatic checking that the pattern is applied correctly.

The goal of the approach described in [6] is to provide a precise description of how pattern participants should collaborate. It states that at the general level, a parameterized collaboration (a UML diagram subtype) is able to represent the structure of the solution proposed by a pattern, but there are severe limitations for which they suggest some UML metamodel modifications.

[8] separates the specification of patterns into three models (role, type and class). The most abstract (role-centric) model presents patterns in their purest

form, capturing their essential spirit. A type-model refines the role-model, and is further refined by a class-model. Further utilizes these ideas in the unambiguous specification of a design pattern.

4. Our Approach

Our main goal is to support software developers in finding the appropriate design patterns based on the informal specification of a pattern. The drawback of the previously mentioned and other existing formalization approaches is that they concentrate only on the UML part (mainly Class Diagrams) of design patterns. Our approach deals with the informal sections of a design pattern. A short summary is created from the textual description of the pattern and stored in a special format. The search method is based on this summary.

In order to realize this goal, the following steps have to be carried out:

- The appropriate format of design patterns has to be elaborated that can also contain this summary.
- The summary has to be created for every pattern based on their informal sections.
- The search method has to be worked out.

As the format of design patterns we chose the pattern metamodel described in PCML (Pattern and Component Markup Language, [10] to be introduced later in this article) and added some mortifications to this metamodel.

The summarization of the patterns has to be done manually. Among the informal parts that we intend to summarize are the *Solution* and the *Consequences* (advantages and disadvantages of applying the solution) sections. Properties of the participants (classes, objects, etc.) of the pattern can also be expressed besides their collaborations, and the way in which they solve a problem. This summary can also be provided when querying the properties of a given design pattern.

The search method can be carried out with a simple algorithm that searches for keywords or any text search engines.

The remaining part of this paper introduces the details of the necessary steps and gives examples how design patterns can be modified to enable search.

5. PCML

Patterns in their original format can be documented with a word processor and some graphical tools. [10] proposes a language called Pattern and Component Markup Language (PCML) which is appropriate for representing design patterns (and components) at any level of abstraction in an XML[2] based format.

[2]Extensible Markup Language

PCML specification provides a standard approach to describe, package, exchange, apply, discover, and extend patterns.

Our approach extends the pattern metamodel described in PCML with two elements called *Predicate* and *Subject*. The predicate element is ordered to the pattern in a one-to-many association while the subject element is related to the predicate in a many-to-many association. These connections enable an arbitrary number of predicates to belong to a design pattern. Neither the number of subjects that belong to a predicate nor the number of predicates that belong to a subject is limited. The modified metamodel is shown in *Fig. 1*.

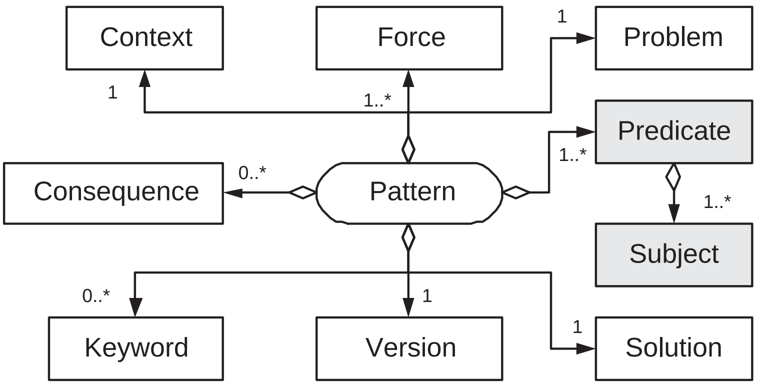

Fig. 1. The modified design pattern metamodel

These two elements are of special interest, namely they encompass what the pattern and its individual objects do and how. They both have a string type attribute which contains the relevant information.

6. Pattern Role Summarization

To make a design pattern suitable for the extended search, the predicate and subject elements have to be prepared. This means that according to the problem description or the intent of a pattern the key verbs and their subjects have to be collected. Besides collecting the verbs and subjects from the original text it is desirable to describe the same problem with other verbs (and other subjects if necessary) too.

This idea is easier to understand through examples so the next subsections demonstrate how design patterns can be completed with the summary.

6.1. Adapter Design Pattern

As the first example we prepare the Adapter [5] design pattern for the search method.

The intent of the pattern is to convert the interface of a class into another interface clients expect. Adapter lets classes work together and objects can be reused that could not otherwise because of incompatible interfaces. This can be the case when the source of the original object is not obtainable.

Use the Adapter pattern when:

- You want to use an existing class, and its interface does not match the one you need,
- You want to create a reusable class that cooperates with unrelated or unforeseen classes, that is, classes that don't necessarily have compatible interfaces,
- (object adapter only) You need to use several existing subclasses, but it's impractical to adapt their interface by sub-classing every one. An object adapter can adapt the interface of its parent class.

Fig. 2 shows the structure of the pattern.

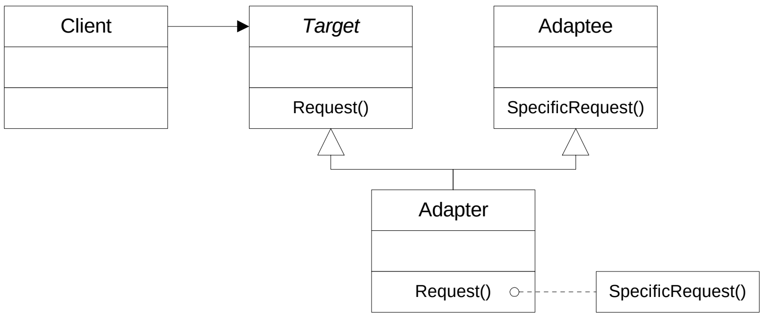

Fig. 2. The structure of the Adapter pattern

The next step is to determine the predicates and their subjects in this pattern. The important predicate and subject pairs are shown in *Fig.* 3. The upper half shows the basic properties of the pattern while the lower half contains more general associations and the advantages of the pattern, e.g. it increases the re-usability of a given class. As these relations are not isomorphic, this graph representation is a good visualization form to show the connections between predicates and subjects.

6.2. *Singleton Design Pattern*

The Singleton design pattern is one of the simplest patterns. The purpose of this pattern is to ensure that a class has only one instance, and to provide a global point of access to it. *Fig.* 4 shows the structure of the pattern.

The collected predicates and subjects are listed in *Fig.*5.

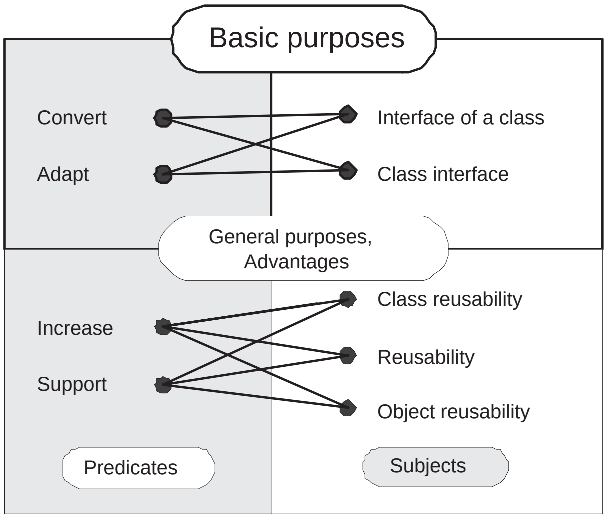

Fig. 3. Adapter - Predicate and Subject pairs

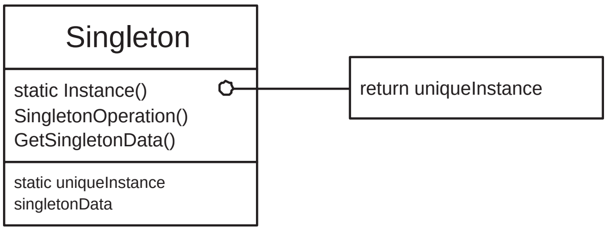

Fig. 4. The structure of the Singleton pattern

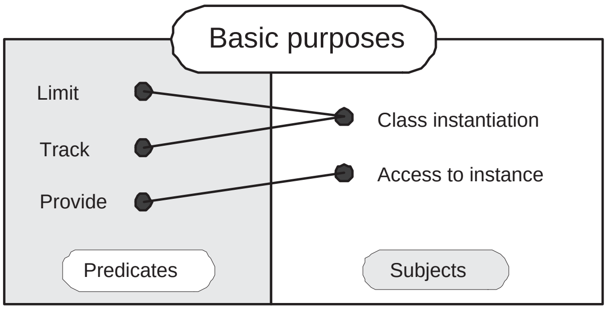

Fig. 5. Singleton - Predicate and Subject pairs

7. Conclusion

With the concept introduced in this paper the developers do not have to know the tremendous amount of design patterns, they simply have to write special search queries according to a system specification, and the appropriate design patterns can be found.

7.1. Advantages of the Approach

When we are looking for design patterns for a system to be developed, we can start with search engines over the Internet. But this way, even if we have typed in the correct keywords, the search can yield inappropriate results as the verbs (predicates) and subjects are not matched. The biggest advantage of our approach is that the verbs and subjects are paired. Several verbs can belong to a design pattern (or any class and object of it) and several subjects can be ordered to every verb and vice versa.

The second advantage of our approach is the ability to query properties of a design pattern or any class and component inside a pattern in a standard way.

When a big amount of patterns are processed the way described in the article, the support for developers to find the right building blocks of a system can be significant.

7.2. Disadvantages of the Approach

The biggest disadvantage of this approach is that the process of collecting the properties of a pattern and the collaborating classes has to be done manually. This means extra work at the documentation of newly discovered design patterns and this work has also to be done in the case of existing, already documented design patterns. In the latter case the processing covers reading and understanding the pattern description and collecting the important keywords.

References

[1] ALEXANDER, C., *The Timeless Way of Building*, Volume 1, Center for Environmental Structure Series, Oxford University Press, New York, 1979.

[2] ALEXANDER, C. – ISHIKAWA, S. – SILVERSTEIN, M., *A Pattern Language*, Volume 2, Center for Environmental Structure Series, Oxford University Press, New York, NY, 1977.

[3] COPLIEN, J. O. – SCHMIDT, D. C., *Pattern Languages of Program Design*, Addison-Wesley, Reading, MA, 1995.

[4] EDEN, A. H., LePUS: A Visual Formalism for Object-Oriented Architectures, *The 6th World Conference on Integrated Design and Process Technology*, Pasadena, California, June 26-30, 2002.

[5] GAMMA, E. – HELM, R. – JOHNSON, R. – VLISSIDES, J., *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley Publishing Company, Addison-Wesley Professional Computing Series, New York, NY, 1977.

[6] LE GUENNEC, A. – SUNYÉ, G. – JÉZÉQUEL, J.-M., Precise Modeling of Design Patterns, eds.: Evans, A. – Kent, S. – Selic, B., UML 2000 – The Unified Modeling Language. Advancing the Standard. Third International Conference, York, UK, October 2000, Proceedings, Vol. 1939 of LNCS, Springer, 2000, pp. 482–496.

[7] HEDIN, G., Language Support for Design Patterns Using Attribute Extension, *Lecture Notes in Computer Science*, Springer, Vol. 1357 (1998), pp. 137–153.

[8] LAUDER, A. – KENT, S., Precise Visual Specification of Design Patterns, *Lecture Notes in Computer Science*, Vol. 1445 (1998), pp. 114–136.

[9] Object Management Group, *Object Constraint Language*, 2002.

[10] Object Venture Inc., *Pattern and Component Markup Language*, 2002, http://www.objectventure.com/pcml.html.

[11] VLISSIDES, J. M. – COPLIEN, J. O. – KERTH, N. L., *Pattern Languages of Program Design*, Vol. 2., Addison–Wesley, Reading, MA, 1996.