

Analyzing Object-Oriented Design Patterns from an Object-Process Viewpoint

Galia Shlezinger¹, Iris Reinhartz-Berger², and Dov Dori¹

¹ Faculty of Industrial Engineering and Management,
Technion-Israel Institute of Technology, Haifa 32000, Israel
galias@tx.technion.ac.il, dori@ie.technion.ac.il
² Department of Management Information Systems,
University of Haifa, Haifa 31905, Israel
iris@mis.haifa.ac.il

Abstract. Design patterns are reusable proven solutions to frequently occurring design problems. To encourage software engineers to use design patterns effectively and correctly throughout the development process, design patterns should be classified and represented formally. In this paper, we apply Object Process Methodology (OPM) for representing and classifying design patterns. OPM enables concurrent representation of the structural and behavioral aspects of design patterns in a single and coherent view. Comparing OPM and UML models of seven popular design patterns, we found that the OPM models are more compact, comprehensible and expressive than their UML counterparts. Furthermore, the OPM models induce a straightforward classification of these design patterns into four groups: creational, structural composition, wrapper, and interaction design patterns.

1 Introduction

Design Patterns describe generic solutions for recurring problems to be customized for a particular context. They have attracted the interest of researchers and practitioners as proven reusable solutions to frequently occurring design problems. Shalloway and Trott [19] suggested several reasons for using, studying, and dealing with design patterns, including the reuse of existing, high-quality solutions and establishing common terminology to improve communication within teams. However, deploying these solutions to develop complex information systems is a tedious task that involves integration issues and iterative development. It is, hence, important to describe design patterns, the problems they intend to solve, the context in which they can be reused, and their consequences in a formal, unambiguous way that can be easily understood by designers. The idea of identifying and reusing the common characteristics of a problem has also been adapted in other fields; one such example is generic tasks [4] in knowledge-based systems.

The increasing number of design patterns that have been proposed and published over the last decade emphasizes the need for a good design patterns representation language, as well as a framework for organizing, classifying, categorizing, and evaluating design patterns, which will help designers in choosing and implementing the

correct solutions to the problems at hand. In this work, we suggest Object Process Methodology (OPM) [6] for both purposes. OPM, which supports two types of equally important elements, objects and processes, enables the representation of both structural and behavioral aspects of design patterns in a single coherent view. Furthermore, the OPM design pattern models lend themselves naturally to a clear, useful classification. This classification is directly derived from the OPM models and does not require justifications and further explanations in plain text. The contribution of our work is therefore two-fold: First, we apply OPM to model and portray the essence of design patterns in a more direct, complete, and comprehensible way than what can be done using object-oriented languages. The completeness of the pattern models is due to OPM's ability to represent both the structure and the behavior of the patterns. Secondly, the categories of design patterns defined in this work are solidly grounded by distinctive characteristics of their respective OPM models. Based on this classification, design patterns can be used regardless of the chosen modeling language.

The rest of the paper is structured as follows. In Section 2 we review work relevant to design pattern languages and classification frameworks. Section 3 provides a short overview of OPM, while Section 4 presents OPM models of several design patterns, making the point that these models induce an accurate design pattern classification scheme. Section 5 concludes our work.

2 Motivation and Background

As noted, design patterns provide solutions for recurring design problems. The idea of documenting design solutions as patterns is attributed to the American architect Christopher Alexander [1]. Applying his idea to object-oriented design and programming, design patterns are usually described using a template. The template used by Gamma et al. in [11], for example, consists of *pattern name and classification, intent (motivation), applicability, structure, participants, collaboration, consequences, implementation, sample code, known uses, and related patterns*. The structure of the design pattern is often illustrated by a graphical representation, such as such as OMT [18] or UML [14] (in this work we will not differentiate between OMT and UML class diagrams). Sometimes, the representations are also accompanied by brief textual descriptions of the basic elements (i.e., classes) that compose the design pattern. The behavioral aspect of the design pattern usually gets much less attention, and is sometimes described informally in text or through partial diagrams that specify the collaboration between the various elements. Such semi-formal representations of design patterns hinder their rigorous, systematic use.

Different languages have been proposed to formally represent design patterns. Some employ mathematical descriptions [8], which require a fair amount of mathematical skills and are not easily understood by designers, while others suggest visual representations or markup languages. Several languages for describing design patterns are based on UML or its extensions, e.g., [9, 12]. Their main shortcoming is the lack of formality. Generally speaking, consistency and integrity are Achilles' heel of UML [15, 17]. Markup languages, such as XML and OWL [5, 16], are also used for describing design patterns. While this approach may be very useful for building tools

that support design patterns, markup languages are machine-understandable at the expense of being cryptic to humans [7].

Since the number of design patterns is continuously increasing, there is a growing need not only to formally describe the different aspects of design patterns, but also to organize and classify them according to their purpose or structure. Noble [13] and Zimmer [20], for example, have classified relationships among design patterns that are mostly hierarchical. Zimmer's categorization brought him to the conclusion that the Factory Method design pattern is not really a pattern, but rather a manifestation of a "X uses Y" relationship between two other design patterns: Abstract Factory and Template Method. He also suggested modeling behaviors as objects and introduced a new design pattern, called Objectifying Behavior, for this purpose.

Gamma et al. [10, 11] have used two dimensions for classifying design patterns: *scope*, which specifies whether the pattern applies to classes or objects, and *purpose*, which reflects the intension of the patterns. According to the purpose dimension, a design pattern can be creational, structural, or behavioral.












Another design pattern classification scheme [2] organizes patterns according to their granularity, functionality, and structural principles. According to this categorization, design patterns are only one group of patterns that belong to a specific level of granularity. Design patterns were further divided into *structural decomposition*, *organization of work*, *access control*, *management*, and *communication* [3].

Since the classification schemes of design patterns are basically object-oriented, their categorization, justified primarily with objects in mind, may not be comprehensive. Adopting Object Process Methodology (OPM), which departs significantly from the object-oriented paradigm, our approach to modeling and categorizing design patterns is fundamentally different. These differences are most evident in design patterns that involve dynamic aspects, since in OPM structure and behavior are equally important. Analyzing the object-oriented classification of the design patterns in [11] with respect to their OPM design pattern models, we offer an improved classification scheme.

3 Object-Process Methodology

Object Process Methodology (OPM) is an integrated modeling paradigm to the development of systems in general and information systems in particular. The two equally important class types in OPM, objects and processes, differ in the values of their perseverance attribute: the perseverance value of object classes is static, while the perseverance value of process classes is dynamic. OPM's combination of objects and processes in the same single diagram type is intended to clarify the two most important aspects that any system features: structure and behavior. OPM supports the specification of these features by structural and procedural links that connect things. Structural links express static relations such as aggregation-participation and generalization-specialization between pairs of things. Procedural links, on the other hand, connect things to describe the behavior of a system, i.e., how processes transform, use, and are triggered by objects. More about OPM can be found in [6]. Table 1 summarizes the OPM elements used in this paper.

Table 1. Main OPM elements, their symbols, and semantics

| Element Name | Symbol | Semantics |
|-------------------------------|---|---|
| Object |  | A thing that has the potential of unconditional existence |
| Process |  | A pattern of transformation that objects undergo |
| Instrument link |  | A procedural link indicating that a process requires an unaffected object (input) for its execution |
| Effect link |  | A procedural link indicating that a process changes an object |
| Result link |  | A procedural link indicating that a process creates an object |
| Event link |  | A procedural link indicating that a process is activated by an event (initiated by an object) |
| Invocation link |  | A procedural link indicating that a process invokes another process |
| Structural Link |  | A structural relation between objects |
| Aggregation-Participation |  | A structural relation which denotes that a thing (object or process) consists of other things |
| Exhibition-Characterization |  | A structural relation representing that a thing (object or process) exhibits another thing |
| Generalization-Specialization |  | A structural relation representing that a thing is a sub-class of another thing |

4 Classification of Design Patterns

In this section we examine the classification of design patterns presented in [11] in terms of OPM. Some of the patterns that are discussed there are modeled using UML class and sequence diagrams. Due to lack of space, we present here only the UML class diagrams of the design patterns.

4.1 Creational Design Patterns

Creational design patterns relate to class instantiation. Figures 1 and 2 describe two creational design patterns: *Factory Method* and *Builder*. Each description includes a problem definition, suggested solution, and UML and OPM models.

As the models in these two figures demonstrate, the UML and OPM models of the design patterns differ in both orientation and abstraction level. While the UML models are object-oriented, i.e., they comprise object classes only, the OPM models are composed of both object and process classes. Thanks to the notion of stand-alone processes, the OPM design pattern models do not require supplementary object classes, which are used only as owners of methods (or operations) or as containers of other classes.

Furthermore, OPM enables specification of behaviors. The UML model of the Factory Method design pattern, for example, requires specification of calling the "Factory Method" from "An Operation" and indicating by informal notes that the "Factory Method" returns a "Concrete Product". The OPM model, on the other hand, specifies these requirements formally by using multiplicity constraints (zero to many, as discussed next) on the number of operations before and after the "Factory Method", and a result link, indicating that the "Factory Method" generates and returns a "Product". Multiplicity constraints in OPM can be associated not only to relations but also to object and process classes. A multiplicity constraint on a thing (object or process) in a design pattern model indicates how many occurrences of this thing can appear in an application that implements the design pattern. In the case of the Factory method design pattern, the process "Operation" can appear zero or more times, as indicated by the "0..n" label at the upper left corner of the process, implying that "Factory Method" is called from somewhere within the process called "An Operation", including its very first or last operation.

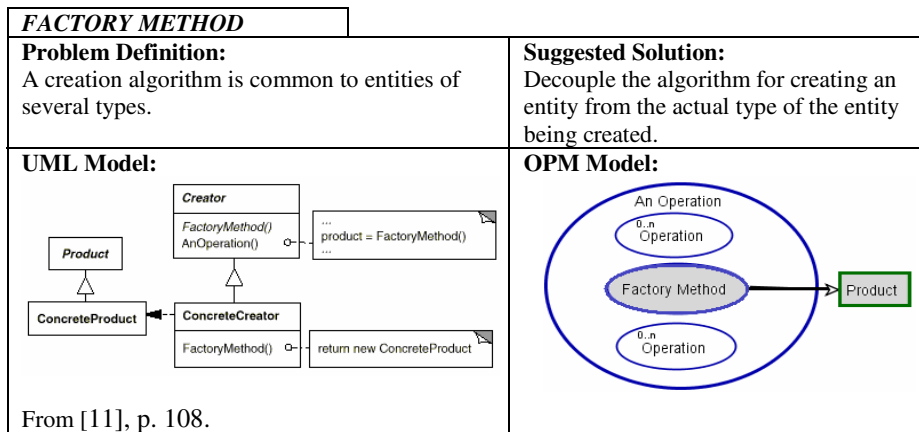


Fig. 1. The Factory Method design pattern

These models also demonstrate the scaling mechanisms that are built into OPM for enabling abstraction and refinement of things. These mechanisms enable detailing an OPM model without losing the "big picture" of the system or pattern being modeled. The mechanism used in this case is in-zooming, in which an entity is shown enclosing its constituent elements. The vertical axis is the time line, so within an in-zoomed process it defines the execution order of the subprocesses, such that subprocesses that need to be executed in a sequence are depicted stacked on top of each other, with the earlier process on top of a later one. "An Operation" is in-zoomed here to display its subprocesses: first a set of zero or more "Operations" is performed, then the "Factory Method" is activated (creating "Products"), and finally another set of 0 or more "Operations" is performed.

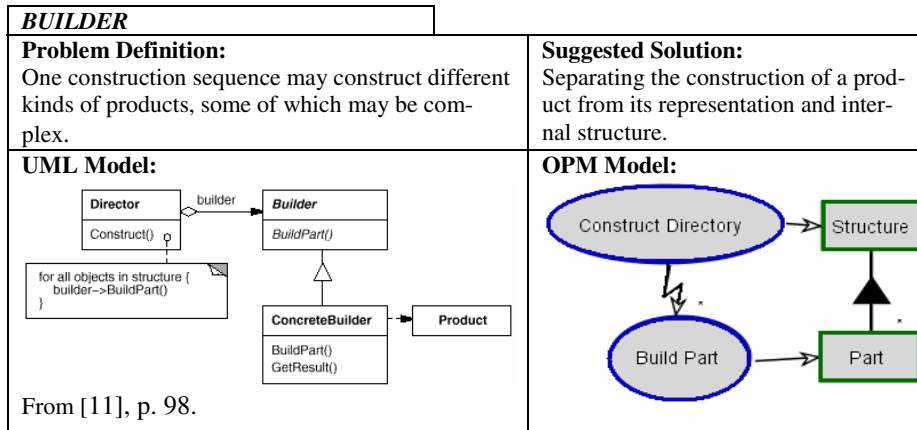


Fig. 2. The Builder design pattern

The second difference between the UML and OPM design pattern models is in their support of abstraction levels. While in the UML models both the abstract and concrete classes appear in the models, in the OPM models, which are actually meta-models, only the "abstract" classes appear, while the concrete classes that implement the abstract ones appear only in the actual models that make use of the design pattern. In these concrete application models, the application elements will be classified according to the design pattern elements using a mechanism similar to UML stereotyping. This separation of the design pattern model—the OPM metamodel—from the application model results in a more abstract, formal, compact, and comprehensible design pattern model. The OPM design pattern models can therefore be considered as templates for applications that make use of these design patterns or as guiding meta-models taken from some meta-library.

Studying the OPM models of the creational design patterns reported in [11] clearly shows that the basic idea behind creational design patterns is separating the construction logic from the objects. The construction logic can be specified as a process that creates the required object(s) as denoted by a result link from the process to the object. This recurring **process – result link – object** pattern is distinguishable in the OPM models in figures 1 and 2 by the pertinent object and process being marked in grey and with thick lines. This pattern indeed justifies the classification of these two design patterns as creational.

4.2 Structural Design Patterns

Structural design patterns relate to class and object composition. They use inheritance to compose interfaces and define ways to compose objects to obtain new functionality. Figures 3 and 4 respectively describe the *Decorator* and *Composite* structural design patterns.

The design patterns in this group further emphasize and clarify the fundamental differences between the UML and OPM design pattern models: All the constraints which are specified in the UML models as notes (in plain text) are expressed formally

in the OPM models. The Decorator design pattern in figure 3, for example, requires adding behavior to a "component". This addition is specified in the UML model as a note which the "Decorator" "Operation" (which includes the "Component" "Operation") is first called and then followed by the "Added Behavior". However, this is only one possibility defined in the design pattern problem section, which reads: "There is a need to add functionality that *precedes or follows* a basic functionality..." The OPM model enables any number of operations *before and after* the basic functionality, which is called in the model "Component Operation".

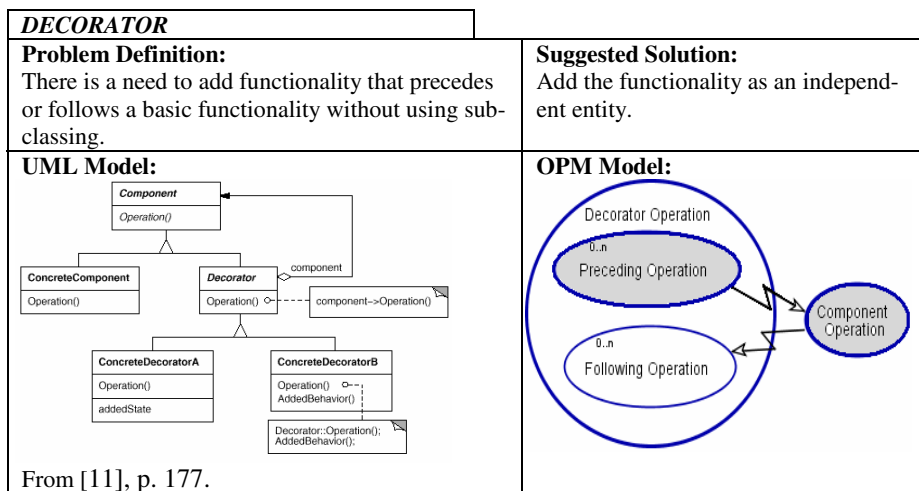


Fig. 3. The Decorator design pattern

The UML and OPM models of the Composite design pattern in figure 4 are completely different. The UML model uses an aggregation relation between "Composite" and "Component", while in the OPM model, "Composite" is zoomed into "Components". However, zooming into an OPM process has both structural aspects (containment, part-whole relations), and procedural aspects (execution order). Furthermore, we have found that the inheritance relation between "Composite" and "Component" in the UML model of this pattern is redundant in the OPM model, since its only meaning is that they are both processes.

The recurrent pattern in the OPM models of this group of design patterns is **process – invocation link – process**. Another observation is that all the design pattern models contain a process which is further zoomed into subprocesses, one of which invokes another process. Although on the surface this pattern should be classified as behavioral rather than structural, a deeper look at the OPM models shows that they basically define the structure of components, each of which is a process. The OPM models of the Factory Method and Decorator design patterns reinforce this observation. The two design patterns are similar in that they both have an operation (performing some function) that may be preceded and/or followed by any number of other

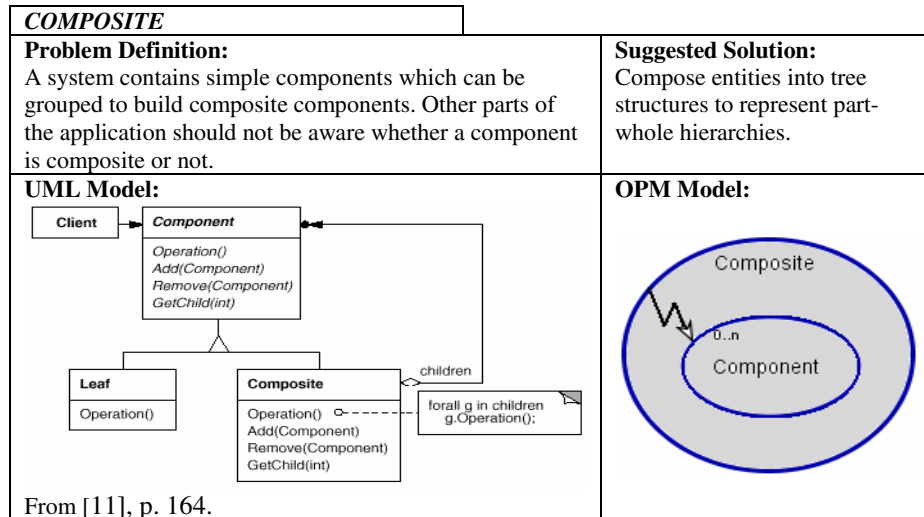


Fig. 4. The Composite design pattern

operations. However, the Factory Method emphasizes the creation of a product, while the Decorator focuses on the separation between the basic functionality—the "Component Operation" and the operation that uses it—the "Decorator Operation".

The OPM model of the Decorator design pattern represents a structural aspect which is common to many behavioral patterns: the invoked process is external to the in-zoomed process. This common aspect justifies their classification by several design pattern classification schemes, including [10], [11], and [19], as *wrappers*. The wrapping essence of the Decorator design pattern is supported by its OPM model, in which one process actually wraps a call to another process.

4.3 Behavioral Design Patterns

Behavioral design patterns define algorithms and object responsibilities. They also help in designing communication modes and interconnections between different classes and objects. Figures 5-7 describe three behavioral design patterns: *Chain of Responsibility*, *Observer*, and *Template Method*.

As the OPM models of these behavioral design patterns clearly show, the focus here is on behavior and way of invocation. It should, hence, come as no surprise that most of the OPM models in this category are dominated by processes. In the OPM model of the Chain of Responsibility design pattern, a "Handler" invokes (triggers) its successor. In the OPM model of the Observer design pattern, there are two different processes: "Notify", which is triggered by the "Subject" and affects the relevant "Observers", and "Update", in which the "Observer" can change the state of the "Subject".

The pattern that characterizes most of the behavioral design pattern OPM models is **object – event link – process – effect link – object**, implying that these design patterns have both triggering and affecting aspects. The two exceptions to this

| CHAIN OF RESPONSIBILITY | |
|--|---|
| <p>Problem Definition: There is a need to invoke a behavior without specifying which implementation should be used.</p> | <p>Suggested Solution: Create a chain of behaviors and pass our request along this chain until it is handled</p> |
| <p>UML Model:</p> <p>From [11], p. 225.</p> | <p>OPM Model:</p> |

Fig. 5. The Chain of Responsibility design pattern

| OBSERVER | |
|---|---|
| <p>Problem Definition: There is a one-to-many dependency relation between objects in the system.</p> | <p>Suggested Solution: Separate between the true object and its observers.</p> |
| <p>UML Model:</p> <p>From [11], p. 294.</p> | <p>OPM Model:</p> |

Fig. 6. The Observer design pattern

characterizing pattern are Chain of Responsibility and Template Method. When modeling the Chain of Responsibility design pattern in OPM, for example, we get a structure of processes that is quite similar to the Decorator structure.

The OPM model of the Template Method design pattern uses the notation of an environmental process (dashed border lines). An environmental thing (object or process) in OPM is either external to the system (pattern) or is an abstract, under-specified thing that needs further specification in the target application model. The OPM model of the Template Method design pattern specifies that the "Template Method" consists of "Internal Operations", as well as "Primitive Operations" that should be further specified in the context of an application. This model is quite similar to the OPM model of the Factory Method; they both define functionality which should be embedded in a behavior

and can be preceded and/or followed by different operations. However, the focus of the Factory Method design pattern is behavioral—the embedded functionality creates products, while the emphasis of the Template Method design pattern is structural—the template consists of internal operations, as well as external (environmental) ones.

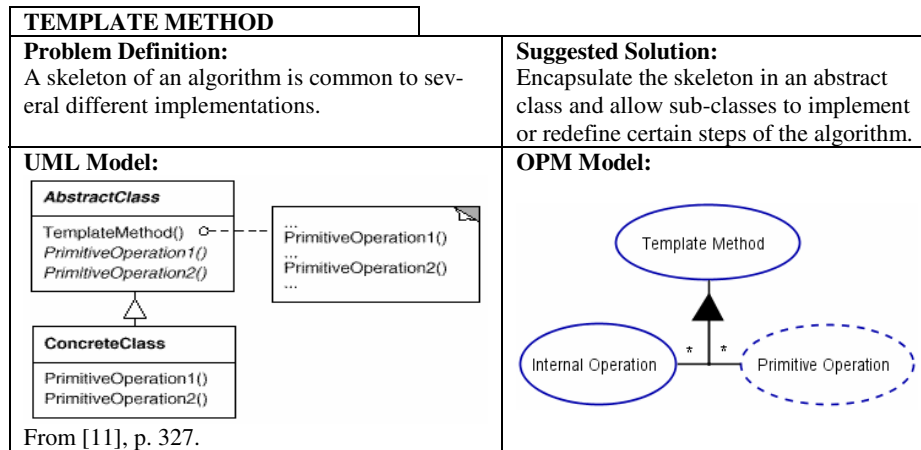


Fig. 7. The Template Method design pattern

4.4 Classifying Design Patterns from an OPM Viewpoint

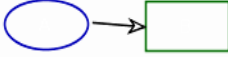
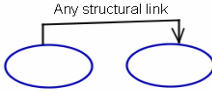
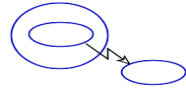
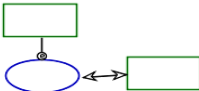
As we have seen, the OPM models of the design patterns reviewed in this paper induce a refined and precise way to classify design patterns. This classification includes four groups, listed in Table 2. The first group of *creational design patterns*, which is quite identical to the creational group in [11], is characterized by the OPM pattern of process – result link – object, which purely conveys the idea of a process creating an object.

The second group, *structural composition design patterns*, has the most abstract characterizing structure: two processes are connected via a structural relation of any type. In the Composite design pattern, for example, the in-zooming of "Composite" into "Component" reveals an aggregation relationship. Note that in the case of Chain of Responsibility, one process, "Handler", plays the role of both processes. Furthermore, the Chain of Responsibility design pattern is often used together with the Composite design pattern. As their OPM models show, these design patterns are very similar and belong to the same category.

The patterns in the third group are classified as *wrapper design patterns*, since they solve their stated problems by wrapping the original functionality.

Finally, the fourth group is the *interaction design patterns* group. The patterns in this group focus on the interaction between static and dynamic aspects of the solution. In OPM terms, these patterns emphasize procedural links, namely effect and event links, which are responsible for updating objects and triggering processes, respectively. This common structure can be observed in different manifestations in the OPM models of the different design patterns that belong to this group. In the OPM model of

Table 2. Classification of Design Patterns according to OPM models

| Design Pattern Category | Design Pattern Examples | Typical OPM Model Core |
|-------------------------|---|--|
| Creational | Factory Method Builder |  |
| Structural composition | Chain of Responsibility Composite Template Method |  |
| Wrapper | Decorator |  |
| Interaction | Observer |  |

the Observer design pattern, the characteristic structure of the interaction design patterns appears twice.

5 Conclusions and Future Work

To encourage software engineers to employ design patterns throughout the entire software development process, the design patterns should be classified logically and represented formally, so their retrieval would be effective and their usage—correct. Since different stakeholders engaged in systems development are more likely to remember visual representations of ideas than textual ones, both UML and OPM suggest ways to model design patterns graphically.

In this paper, we have presented OPM models of seven popular design patterns. We pointed out how the OPM models of the design patterns convey the essence of the solutions offered by the patterns and how these OPM models help designers integrate them into their application models. Comparing the design patterns' OPM models to their UML counterparts, we have shown that the former are more expressive and formal. Furthermore, we found out that the OPM models induce a logical classification of the design patterns into four groups: creational, structural composition, wrapper, and interaction. This classification refines the classification in [11] and identifies some problems in the categorization there (Chain of Responsibility and Template Method, for example, should be categorized as structural design patterns rather than behavioral ones).

We plan to apply our approach to additional design patterns and develop models of complete applications that host design patterns. We also plan to develop ways to retrieve design patterns easily (using OPM models) from the problem domain.

References

1. Alexander, C., Ishkawa, S., Silverstein, M., Jacobson, M., Fiksdahl-King, I., Angel, S.: A Pattern Language. New York: Oxford University Press (1977).
2. Buschmann, F., Meunier, R.: A System of Patterns. In: Coplien, J. O. and Schmidt, D. C. (eds.): Pattern Language for Program Design, Addison-Wesely (1995), pp. 325-343.
3. Buschmann, F., Meunier R., Rohnert, H., Sommerland, P., Stal, M.: Pattern-Oriented Software Architecture: a System of Patterns. Wiley (1996).
4. Chandrasekaran, B.: Generic tasks in knowledge-based reasoning: high-level building blocks for expert system design. *IEEE Expert* (1986), pp. 23-30.
5. Dietrich, J., Elger, C.: A Formal Description of Design Patterns using OWL. Proceedings of the 2005 Australian software engineering conference (2005), pp. 243-250.
6. Dori, D.: Object-Process Methodology – A Holistic System Paradigm. Springer (2002).
7. Dori, D.: ViSWeb – The Visual Semantic Web: Unifying Human and Machine Knowledge Representations with Object-Process Methodology. The International Journal on Very Large Data Bases, 13, 2, pp. 120-147, 2004.
8. Eden, A. H., Hirshfeld, Y., Yehudai, A.: LePUS – A declarative pattern specification language. Technical report 326/98, department of computer science, Tel Aviv University (1998).
9. France, R. B., Kim, D. K., Ghosh, S., Song, E.: A UML-Based Pattern Specification Technique. *IEEE Transactions on Software Engineering* (2004), 30 (3), pp. 193-206.
10. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Abstraction and Reuse of Object Oriented Design. Proceedings of the 7th European Conference on Object Oriented Programming. Berlin: Springer-Verlag (2003), pp. 406-431.
11. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley (1994).
12. Lauder, A., Kent, S.: Precise Visual Specification of Design Patterns. Lecture Notes in Computer Science (1998), Vol. 1445, pp 114-136.
13. Noble, J.: Classifying relationships between Object-Oriented Design Patterns. Proceedings of the Australian Software Engineering Conference (1998), pp. 98-108.
14. Object Management Group. UML 2.0 Superstructure FTF convenience document. <http://www.omg.org/docs/ptc/04-10-02.zip>.
15. Peleg, M., Dori, D.: The Model Multiplicity Problem: Experimenting with Real-Time Specification Methods, *IEEE Transaction on Software Engineering* (2000), 26 (8), pp. 742-759.
16. Petri, D., Csertan, G.: Design Pattern Matching. *Periodica Polytechnica Ser. El. Eng* (2003). Vol. 46, no. 3-4, pp. 205-212.
17. Reinhartz-Berger, I.: Conceptual Modeling of Structure and Behavior with UML – The Top Level Object-Oriented Framework (TLOOF) Approach. Proceedings of the 24th International Conference on Conceptual Modeling (2005), Lecture Notes in Computer Science 3716, pp. 1-15.
18. Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorenson, W.: *Object-Oriented Modeling and Design*. Prentice Hall, Englewood Cliffs, NJ (1991).
19. Shalloway, A., Trott, J.: Design Patterns Explained: A New Perspective on Object-Oriented Design. Addison-Wesley (2001).
20. Zimmer, W.: Relationships Between Design Patterns. In: Coplien, J. O. and Schmidt, D. C. (eds.): Pattern Language for Program Design. Addison-Wesely (1995), pp. 345-364.