Object-oriented patterns.

by Peter Coad

Object-oriented analysis (OOA) and object-oriented design (OOD) rely on classes and objects as the lowest level building blocks. These classes and objects form patterns with specific relationships between them. Groups of classes in an object-oriented environment are likely to be useful repeatedly. Many patterns may be found by trial-and-error and by observation. Examples of pattern types include item description, time association, event logging, roles played, state across a collection, and broadcast. Patterns standardize small piecework into a larger unit and become building blocks for program design and construction.

This article explores patterns, how to find them, presents some patterns for object-oriented analysis (OOA) and object-oriented design (OOD) as well as providing examples and guidelines for applying them.

Patterns Apply to Many Disciplines

What is a pattern?

Pattern. A fully realized form, original, or model accepted or proposed for imitation: something regarded as a normative example to be copied; archetype; exemplar (10).

Many fields use patterns in various ways: In music and literature, a pattern is the coherent structure or design of a song or book. In art, a pattern is the composition or plan of a work of graphic or plastic art. In architecture, a pattern is an architectural design or style.

In psychology, a pattern is a thinking mechanism that is basic to the brain's operation, helping one to perceive things quickly (1). In archeology, a pattern is a group of phases having several distinguishing and fundamental features in common. In linguistics, a pattern is the manner in which smaller units of language are grouped into larger units.

In dressmaking, a pattern is a pleasing shape that is applied repeatedly. In decorating, a pattern is a design or figure appearing in furniture or an accessory. In manufacturing, a pattern is the shape or style of a manufactured form. In aviation, a pattern is a collection of approaches, turns, and altitudes prescribed for an airplane that is coming in for a landing. In broadcasting, a pattern is a standard diagram for testing television circuits.

In numismatics, a pattern is a specimen of a proposed coin or coin design. In chess, a pattern is a set of moves that may be applied in an overall strategy (10).

With each pattern, small piecework is standardized into a larger chunk or unit. Patterns become the building blocks for design and construction. Finding and applying patterns indicates progress in a field of human endeavor.

How do People Discover Patterns?

In his book The Timeless Way of Building (2), noted architect Christopher Alexander examines the importance of architectural patterns:

...every place is given its character

by certain patterns of events that

keep on happening there.... These

patterns of events are locked in

with certain geometric patterns in

the space. Indeed, each building

and each town is ultimately made

out of these patterns in the space,

and out of nothing else; they [pat

- terns in the space] are the atoms

and molecules from which a build

- ing or a town is made (2).

Patterns are more than just the smallest elements in an endeavor.

On the geometric level, we see certain physical elements repeated endlessly, combined in an almost endless variety of combinations .... It is puzzling to realize that the elements, which seem like elementary building blocks, keep varying, and are different every time that they occur .... If the elements are different every time that they occur, evidently then, it cannot be the elements themselves which are repeating in a building or town; these so-called elements cannot be the ultimate "atomic" constituents of space (2).

To find patterns, what does one look for?

Look more carefully...to find out what it really is that is repeating there.... Beyond its elements, each building [or town] is defined by certain patterns of relationships among the elements.... These relationships are not extra, but necessary to the elements .... The elements themselves are patterns of relationships (2).

And what happens when one finds a pattern? One begins to think with that new building block, rather than with littler pieces.

And finally, the things which seem like elements dissolve, and leave a fabric of relationships behind, which is the stuff that actually repeats itself, and gives the structure to a building or a town (2).

How Can One FInd Patterns for OOA and OOD?

So what is the impact of Alexander's insights on advancing OOA and OOD? Object-oriented methods tend to focus on the lowest-level building block: the class and its objects (3, 4, 5, 9, 11).

Object. A person, place, thing, event, or concept.

Class. A description of a number of objects which have certain likenesses or common traits. (derived from (10))

Classes and objects correspond to Alexander's constantly repeating, lowest-level elements. Patterns of lowest-level elements and relationships between them form a building block for more effective OOA and OOD. To find a pattern among some lowest-level elements (classes and objects), one must look at the relationships between them.

Object-oriented methods already emphasize certain patterns of relationships, including generalization-specialization, whole-part, association, and messaging (3, 4, 9). Such relationships tie the lowest-level building blocks together.

Some have investigated application frameworks (7), a skeleton of classes, objects, and relationships grouped together for building a specific application. Most of these application frameworks are primarily human interaction skeletons, providing a more systematic approach to building window interfaces. Examples include Apple's MacApp (for building a Mac interface), Borland's ObjectWindows, and the model-view-controller architecture within ParcPlace System's Objectworks (6, 8).

But other combinations--ones likely to be applicable multiple times within a single application, and likely to be applicable across many different kinds of applications-have not been investigated to date. Little is known about patterns-combinations of certain classes and objects, with relationships between them--that apply again and again in different OOA and OOD efforts. Exploring such patterns is the purpose of this short article.

An object-oriented pattern is an abstraction of a doublet, triplet, or other small grouping of classes that is likely to be helpful again and again in object-oriented development.

Patterns are found by trial-and-error and by observation. By building many object-oriented models and by observing many applications of the lowest-level building blocks and the relationships established between them, one can find patterns. With such patterns, as Alexander observed, "the things which seem like elements dissolve," and one is able to use a higher-level building block for OOA and OOD.

Seven Patterns, Seven Examples, and Guidelines

This section begins with notation, followed by a presentation of the following patterns: item description; time association; event logging; roles played; state across a collection; behavior across a collection; and broadcast. With each pattern discussed, examples and guidelines are provided. Notation This article uses the notation summarized in Figure 1.

"Item Description" Pattern Figure 2 illustrates the "item description'' pattern.

The pattern. The item description pattern consists of an "item" object (i.e., an object of the class "item") and an "item description" object. An "item description" object has attribute values which may apply to more than one "item" object; an "item" object has its own individual assignment of attribute values.

An example. An "aircraft" object knows its own tail number (e.g., N123ABC); it also knows about exactly one "aircraft description" object. An "aircraft description" object knows its own manufacturer (e.g., Boeing), model (e.g., 747-400), and standard cruising range (e.g., 8,333 miles); it also may know about some number of "aircraft" objects that depend on that information.

Guidelines for use. Use this pattern when some attribute values may apply to more than one object in a class.

"Time Association" Pattern Figure 3 illustrates the "time association" pattern.

Object-oriented patterns.

The pattern. A "participant 1" object may know about (be associated with) a "participant 2" object. If one needs to express attributes or services regarding that association, then an object from "time association" is needed. A "time association" object often sends messages to its participating objects in order to get values or get a subcalculation done on its behalf. Note that the association connection (I) captures the association for future. queries about these objects and (2) captures (for the sender)"to whom to send a message."

An example. A "legal event" object knows its date and time; it also knows about (and ties together, in association) some number of "owner" objects and exactly one "vehicle" object. To calculate a fee, a "legal event" object sends the message "assess tax type" to its corresponding owner object(s) and then sends the message "categorize vehicle" to its corresponding vehicle object. An "owner" object knows its name and address; it also knows about some number of corresponding "legal event" objects. A "vehicle" object knows its number and style; it also knows about some number of corresponding "legal event" objects.

Guidelines for use. Use this pattern whenever the system is responsible to know an association between two or more objects and to know or do something about that association.

"Event Logging" Pattern Figure 4 illustrates the "event logging" pattern.

The pattern. A "device" object monitors an external device; the object is responsible for detecting that an event has occurred; the object is also responsible for initiating the response to that event. Part of that response may be to log the event’s occurrence. When this is the case, a "device" object sends the message "create" to the "event remembered’’ class to create a new object in that class, one with historical values. A "device" object may know about some number of "event remembered" objects; an "event remembered" object must know about a corresponding "device object.

An example. A temperature sensor" object monitors an actual temperature sensor looking for a threshold violation, to do its job, it knows its operational state and its threshold. Once it detects that a threshold violation has occurred, it sends a message to the "threshold violation" class to create a new object in that class with values for date and time, measured value, and monitored threshold.

Guidelines for use. Use whenever an event is detected, and you need to log its occurrence to support after-the-fact analysis or to meet legal requirements.

"Roles Played" Pattern Figure 5 illustrates the "roles played" pattern.

The pattern. A "player" object has attribute values and services that apply over time. A player object is always a player object. At times, a player object "wears different hats," playing one or more roles. Often, starting and ending times are common to all such roles. Roles are specialized, according to the attributes and services needed in each role. This pattern accommodates large numbers of roles, combinations of roles, and changes in roles much more graciously than an application of multiple inheritance permits.

An example. A "video" object knows its name and copy number; it also knows about corresponding "rented video role" and "returned video role" objects. Each role object knows its starting date and time and (eventually) its ending date and time. The "rented video role" knows its duration, and watches for it becoming overdue; the "returned video role" knows its status (i.e., whether or not it is ready to be rented again).

Guidelines for use. Use this pattern whenever you have a player object which remains the same old player object, but has different attributes and services, depending on the "hats" the player may wear. Use this pattern to model large numbers of roles, combinations of roles, and changes in roles; this approach is more concise and flexible than attempting to use multiple inheritance in this situation.

"State Across a Collection" Pattern

Figure 6 illustrates the "state across a collection" pattern.

The pattern. A "collection" object knows its state; this state applies to the collection and may also apply to its parts, by physical or temporal proximity. And each "member" object has its own state, too.

An example. An aircraft is an assembly (collection) of engines; in other words, an "aircraft" object may know about some number of "engine" objects. (Note that while most aircraft have engines, gliders do not.) Each "engine" object knows its own rated power. Each "aircraft" object knows about its altitude; this particular attribute value applies to the whole, and also to its parts, by physical proximity.

Guidelines for use. Use this pattern whenever there is whole-part in a business domain or implementation domain, and one or more attributes apply to the whole (the collection).

"Behavior Across a Collection" Pattern

Figure 7 illustrates the "behavior across a collection" pattern.

The pattern. A "collection" object has behavior that applies across an entire collection of its "member" objects. And each "member" object performs actions, knowing (by means of its attributes) how to perform, without needed coordination with other "member" objects.

An example. A "call" object knows its time of arrival, priority, and its originating number. The "call" object (the abstraction, not the electronic signal it works on) also knows how to route itself. It can even rate its importance.

But which call gets to go next? The "call" object does not know enough to make the actual selection. Yet a collection of all calls, called a "call collection" object in this example, can know enough (all the call objects waiting to be serviced) and do enough (select the next call, based on a selection algorithm) to carry out this responsibility. So the "call collection" object is set up to do just that. Note that such a collection does exist in the domain; it is called a queue; but the words "call collection". take the model away from being locked into a "first in, first out" mentality; and the name helps one to focus on behavior across the collection only, not pulling up any of the work that each call knows enough to do itself.

Guidelines for use. Use this pattern whenever there is whole-part in a business domain or implementation domain, and a behavior (i.e., one or more services) applies across the whole collection. Caution: make the member objects do as much as they can with what they knoW; only put behavior that really applies across the collection up in the "collection" object; in doing this, the active side of the objects is better partitioned by the participating classes, rather than having a centralized manager with subordinate data-hiders.

"Broadcast" Pattern

Figure 8 illustrates the "broadcast" pattern.

The pattern. This pattern is used to communicate complex changes between one major section of an OOA/OOD model with another major section. Whenever it changes, a "broadcasting item" object broadcasts a change notification to the "receiving item" objects that it knows about. A notified "receiving item" object then sends a message to the "broadcasting item" to get the change; once it gets the change, a "receiving item" object takes whatever action is necessary in light of the change.

An example. On the left side of Figure 8, the pattern is applied to keep human interaction distinct from business domain classes. This is done to simplify both parts; and it is done to increase the likelihood of reuse for each part. A "human interaction view" object gets user input and sends a message to invoke action to the corresponding "model" object. At some point in time, when a change does occur, a "model" object broadcasts a change notification to its dependent "human interaction view" objects. Then each dependent "human interaction view" object sends a message to get the change; on receipt of the change, the "human interaction view" updates its display. In ParcPlace System’s Objectworks, this application of a broadcast pattern is called model-view-controller (MVC); the innovative work on MVC (6, 8) is the basis for abstracting and then applying the "broadcast" pattern.

On the right side of Figure 8, the pattern is used to isolate the impact of data management. Again, this is done to simplify both parts; and it is done to increase the likelihood of reuse for each part. At some point in time, when a change does occur, a "model" object broadcasts a change notification to its dependent "data interaction view" objects. Then each dependent "data interaction view" object sends a message to get the change; on receipt of the change, the "data interaction view" updates its data representation (e.g., its tables). The "data interaction view" knows how to save and load its data representations into a storage device.

Actually, this example is somewhat simplified; often, in the application of this pattern, the interacting objects are actually objects of classes that are specializations of the classes shown in this example. Yet the same basic pattern applies, even when specialization classes are involved.

Guidelines for use. Use this pattern to establish interactions between major OOA/OOD parts in a way that the two sections stay cleanly separated, rather than becoming hopelessly intertwined. Be sure to use this pattern to separate business domain classes from human interaction classes, and to separate business domain classes from data management classes.

Applying SIx Patterns In One Example

Figure 9 is a larger example showing how patterns can be combined into larger models. This one model applies six patterns. summary and

Recommendations

A pattern is a fully realized form original, or model accepted or proposed for imitation. With patterns, small piecework is standardized into a larger chunk or unit. Patterns become the building blocks for design and construction. Finding and applying patterns indicates progress in a field of human endeavor.

This article is only a very small beginning of the work to be done on investigating, finding, and applying object-oriented patterns. Additional investigation is needed on pattern discovery and usage. Given a large number of OOA and OOD results, can one apply a systematic approach to discovering and cataloging patterns? Is there a hierarchy of patterns? How does one look at examples and derive guidelines for best usage? What strategies can be used for connecting one pattern to another? When does the occurrence of one pattern imply the need for another companion pattern?

Patterns are the molecules from which one may apply OOA and OOD more effectively.

Acknowledgments The author thanks Mark Mayfield, Jill Nicola, and Teri Roberts for their help in identifying, abstracting, and applying these patterns. If

References

1. Albrecht, K. Brain Power. Prentice Hall, Englewood Cliffs, N.J., 1980.

2. Alexander, C. The Timeless Way of Building. Oxford University Press, 1979.

3. Booch, G. Object-Oriented Design with Applications. Benjamin/Cummings, Redwood City, Ca., 1991.

4. Coad, P. and Yourdon, E. Object-Oriented Analysis. Second ed. Prentice Hall, Englewood Cliffs, N.J., 1991.

5. Coad, P. and Yourdon, E. Object-Oriented Design, Prentice Hall, Englewood Cliffs, N.J., 1991.

6. Goldberg, A. Information models, views, and controllers. Dr. Dobb's J. (July 1990).

7. Johnson, R. and Wirfs-Brock, R. Object-oriented frameworks, Tutorial notes. In Proceedings of ACM OOPSLA (1991),

8. Leibs, D. and Rubin, K. Reimplementing model-view-controller. The Smalltalk Report (Mar./Apr. 1992).

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

10. Webster's Third New International Dictionary. Merriam Webster, Inc., 1986.

11. Wirfs-Brock, R., Wilkerson, B. and Wiener, L. Designing Object-Oriented Software, Prentice Hall, Englewood Cliffs, N.J., 1990.

CR Categories and Subject Descriptors: D.2.1 [Software]: Software Engineering -- requirements / specifications; D. 2.10 [Software]: Software Engineering-design; I.6.0 [Computing Methodologies]: Simulation and Modeling-general; I.6.3 [Computing Methodologies]: Simulation and Modeling-- applications; K.6.3 [Computing Milieux]: Management of Computing and Information Systems--software management; K.6.4 [Computing Milieux]:Management of Computing and Information Systems--system management

General Terms: Design, Experimentation

Additional Key Words and Phrases: Analysis, design, object-oriented notation and methodology, object-oriented software engineering, reliable component reusability

About the Author:

PETER COAD is the chair of Object International, Inc. Current research interests include object-oriented methods, reuse, more effective analysis and design tools, and intelligence theories leading to accelerated technology transfer. Author's Present Address: Object International, Inc., 8140 N. MoPac Expressway, Building 4, Suite 200, Austin, TX 78759-8864; email: coad@ applelink.apple.com or CompuServe 71210, 3642