Using Object Oriented Design Patterns to Develop an Interactive Command System for a CAD Software with Undo and Redo Support

Mehmet D. AKIN, Nadia ERDOĐAN
Istanbul Technical University
Electrical-Electronics Faculty
Computer Engineering Department
Ayazaga, 80626, Istanbul TURKEY
mdakin@mam.gov.tr
erdogan@cs.itu.edu.tr

Abstract. *Although little time has passed since its introduction, object oriented design patterns have had a worldwide interest. This paper presents the use of design patterns to develop a flexible and undoable interactive command system for a CAD software, explaining in detail the design issues and specific cases where the need for tuning and modification of patterns has been observed.*

Keywords: *design patterns, interactive command systems, CAD software*

1. Introduction

Object oriented design patterns were first introduced in a book authored by Gamma E., Helm R. Johnson R. and Vlissides J. called “Design Patterns: Elements of Reuseable Object-Oriented Software” [1] and received a worldwide acclamation at the time. The book describes design patterns as "description of communicating objects and classes that are customized to solve a general design problem in a particular context".

Design patterns allow solutions that were developed by experienced designers and programmers to be named and catalogued. One of the short term benefits of design patterns is the fact that as a result of this naming, they act as a common vocabulary to simplify and enhance the dialog between developers. They also enable the design to be discussed at an abstract level without going into details.

Besides being a documentation tool, design patterns can provide flexible and generic solutions to problems. The outcomes of these solutions can be predetermined as to their pros and cons.

This paper presents a mechanism developed to handle requests of a CAD software. A flexible, pluggable and undo/redo-able command and toolbox system is defined. The following sections introduce the details of desing and implementation phases with the modifications on the patterns used to meet the requirements of the system.

2. Representation of Visible Objects

CAD like interactive drawing programs usually have primitive and compound shaped objects. Primitive objects are lines, points, circles, arcs, polygons and simple text. Compound objects are any set of these primitive objects and they are defined according to the needs of particular contexts. Compound objects may also be named as group or composite objects. Its best to use the *"Composite"* design pattern to represent the group shape objects to achieve a good result (Figure 1). Each shape class is derived from an abstract Shape class and they override methods like "draw" to provide functionality that differs from one shape to another. Rotate, scale, move and other attribute changing methods can also be defined in the Shape base class and overriden in the child classes.

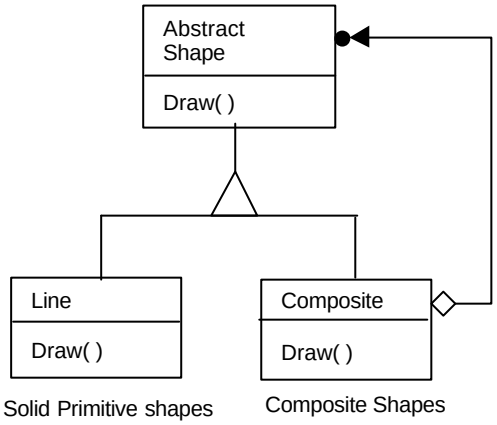

Figure 1. "Composite" Design Pattern

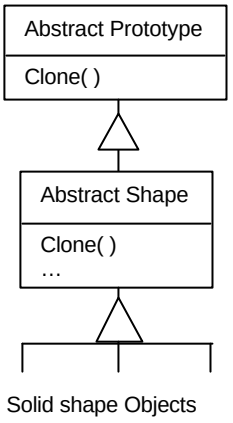

Figure 2. "Prototype" Design Pattern

For a flexible system, we may not know the future needs of the user and we need to allow developers to add new primitive and compound shape objects. The object system must be flexible to allow this. We have used a shape manager and kept object class definitions and prototypes in it. So if a new shape class is to be added, it is registered to the shape manager and its "clone" method is used to create an instance of new shape class (Figure. 2). The *"Prototype"* [1] design pattern is applied here to allow for the control of a centralized and flexible object creation system. The object manager has a unique instance in the system and it must have a global access point, so we designed it as a singleton [1], using the *Singleton* design pattern.

3. Undo and Redo Operations

A complete CAD system should support reliable, unlimited and fast undo-redo operations. Undoing an operation can be defined as reversing the operation. For

operation of creating an object, undoing the operation results in deletion of the created object. Redoing an operation is to re execute the last command that has been undone, for our example, to recreate the deleted object.

There are different kinds of undo-redo mechanisms used in systems. Some software allow the user to undo only one operation. Some systems have a limited undo operation depth using a history buffer. To achieve an unlimited undo and redo depth, the designer should utilize undo files on disk, therefore the only limitation to the depth of undo operation is the disk capacity. Creating thousands of objects in an operation may result in megabytes of data to be transferred during undoing and redoing the operation.

In CAD applications, most of the operations change the internal state of the object structure of the system. Undo operation recalls the previous states of the system, and redo does the reverse. But if one undoes several operations and then makes a single operation that changes the state of the system, then the previous operations can not be redone. We call this as a "redo cancellation"

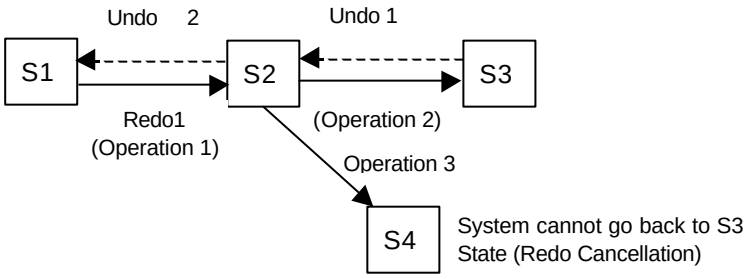

Figure 3. State transitions during undo and redo operations and redo cancellation

As seen in Figure3., the user first executed Operation1 and Operation2. Then thought that Operation1 was not correct and undone both Operations (Undo1 and Undo2). But after a while he realized that Operation1 was correct and redid it. At this point, if the user does not redo Operation2 and instead, executes another operation (Operation 3) he cannot go back to state S3 again.

Before explaining implementation details about the undo-redo handling mechanism of the system, a look at the *"Command"* design pattern and the representation of operations and tools in the CAD system should be appropriate.

4. Command Design Pattern: Representing Operations and Tools

The command design pattern is used to define requests as objects and to support undoable operations[1]. Command pattern encapsulates a function . A command controller can also be used to store the defined solid command prototypes and

dispatch user events to a Command Processor. The command processor is responsible of accepting service requests and performing undo operations [3]. Figure4 depicts the command controller, command processor, abstract command and solid command classes.

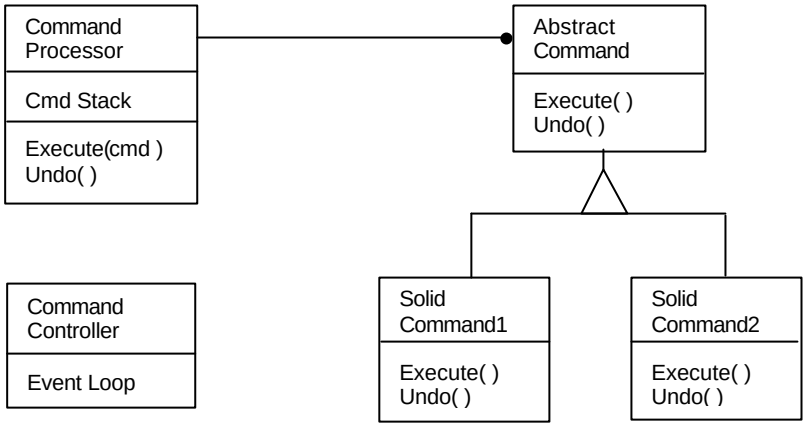

Figure 4. Command design pattern and command processor

Command controller accepts request from a user within an event loop (either selecting from a menu, or using a keyboard shortcut) and creates an appropriate command object for the request and transfers it to Command Processor. Command processor, calls the execute method of command object and stores it in a stack for undoing purposes. Command stores the state information for a possible undo operation and completes its function. When command controller accepts an undo request, its sends it to command processor, which calls the undo method of the command residing on top of the command stack. Command performs the undo method using previously stored state information.

We have observed that this approach for command processing and object manipulation does not completely satisfy the needs of a CAD application for the following reasons.

- The system stores command objects in a stack that limits the undo depth.
- Command object itself is also stored for undoing purposes, this overhead may seem negligible but considering block operations in which thousands of objects attributes are changed , the system may suffer from it.
- The interactive behavior and event handling mechanism of CAD programs are undefined in the command design pattern.
- The system can handle only one command at a time and commands are deleted after their execution. No sub commands or auxiliary commands are defined.

To solve these problems, we need to define another class for the behavior of operations to handle user input. In [1], a brief explanation is given for tool classes that are used for handling user input to manipulate and create objects. We should examine the CAD systems needs carefully to find out a correct solution.

In CAD like interactive systems, interactive operations are divided into two phases. In the first phase, the user selects a tool and creates parameters for the tool interactively using a mouse, keyboard or another input device. For example, the user who chooses the circle tool, first selects coordinates of the center of the circle by clicking the mouse button, and drags the mouse to create a circle with the desired radius. After the second click, the second phase starts; a circle object should be registered to the system using the parameters that have been entered by the user.

These two operations should be separated from each other to solve the problems mentioned above. Vlissides [2] also mentions the separation of roles of classes: one class should be responsible of user input handling, while another class should be responsible of doing work of undo and redo operations. We call the user input handling classes as tools and we can derive them from an abstract tool class.

Basic operations in a CAD system may be grouped into categories. Creating simple, complex or compound objects, selecting objects, modifying attributes of objects and operations to change the visible status of the project like, zooming, panning of canvas or hiding a layer are the widely used categories of operations.

Zoom and pan operations should be done without destroying the previously executed tool. User may wish to zoom into an area to see the details during the creation of a line, or during a selection operation. To cancel a long operation for just zooming or panning is not user friendly, and wastes precious user time. This implies a separation of tool classes as tools and sub-tools.

Another issue about tools is the existence of auxiliary tools for the currently executing tool. For example, during line drawing, we may give the user an opportunity to draw a line the same length as another one: user may press a short cut during the drawing operation and select another line, after which, the first line operation should be finished and a line with the same length as the selected one should be added to system. This implies that we should have a command stack and a messaging mechanism between commands.

Vlissides[2] proposes a system for CAD like interactive systems using *visitor and command* design patterns. He defines a tool class and derives creation, selection, rotation and other tools from this base class, using the tool class as a *"Visitor"*. Visitor helps designers to limit sub-classing. In his work, he defines an abstract tool class and derive other tools from it. Tools are sent to objects and objects send themselves to the tool objects to start their manipulation. Therefore instead of having different creation or modification tools for each different shape, we have one method for each kind of shape in tool classes. This approach reduces sub-classing but does not reduce the amount of code and actually limits the flexibility of adding new shape

objects to system because the visitor design pattern is not appropriate for changing the element hierarchy. One virtual method on the base class for each subclass should be defined and, for each new subclass to be added, a method with a new subclass parameter need also be added.

Therefore, we've chosen not to use the visitor approach, but have defined an abstract tool class and derived all creational and manipulating tools from it. Sub-Tools and Auxiliary tools are also derived from this base Tool class (Figure 5.).

For undo purposes, we have changed the name of the command class and defined an abstract Undo class. All undoable operations are derived from this class. The prototypes of Solid tool objects are kept in the tool manager. Any new tool object can be registered to Tool Manager even during run-time. This exploits the dynamic plug-in capability of system. Any developer, who wants to add new functionality to the CAD kernel, may define a new tool sub class or classes and compile the code with the CAD kernel library provided. Developer should register the classes in the entry function of the dynamicly loadable library. The CAD system either checks the plug-in files during start-up, or even during run-time, new functions and tools can be applied.

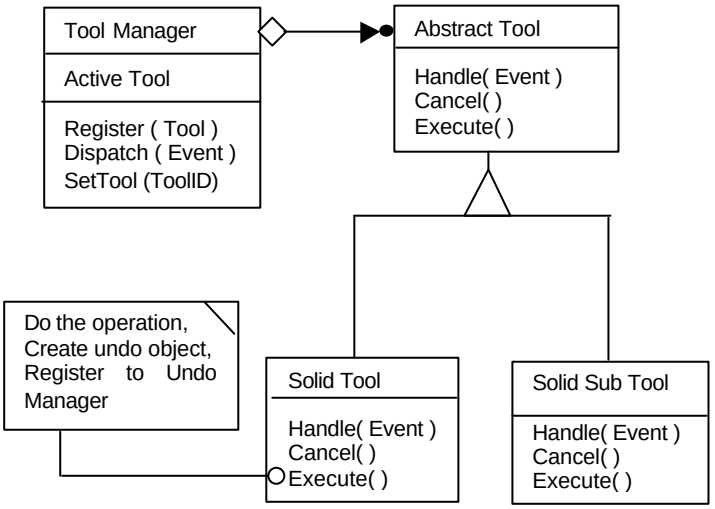

Figure 5. Tool Manager, Abstract Tool, Solid Tool and Solid Sub Tool Classes

The mouse and keyboard events in the system must be forwarded to the tool manager. Tool manager packages the events and sends them to the active tool at the moment, which checks its internal state and executes according to user input. If a sub tool has been chosen, Tool manager checks the Active tool and if it's a normal tool, it pushes it to its stack. After a sub-tool completes its operation, the previous tool is reactivated. Tools also have a cancel option, in case the user wants to cancel the operation (pressing a predefined key like "escape") Tool manager invokes the Cancel method of the active tool. It stops after executing a clean-up operation .

The Undo Manager controls the undo objects. In our CAD kernel, we have an undo buffer in memory, which holds undo objects. When buffer size is exceeded its content are written to a file and another file holds size and position information for each undo object that resides in the file. *"Memento"* design pattern is used here to provide storage of undo objects without violating encapsulation. Undo manager keeps track of both local (in buffer) Undo pointer and a Global (Disk File and buffer) Undo object pointer, to keep track of the object to which the next undo operation will be applied. (Figure 6)

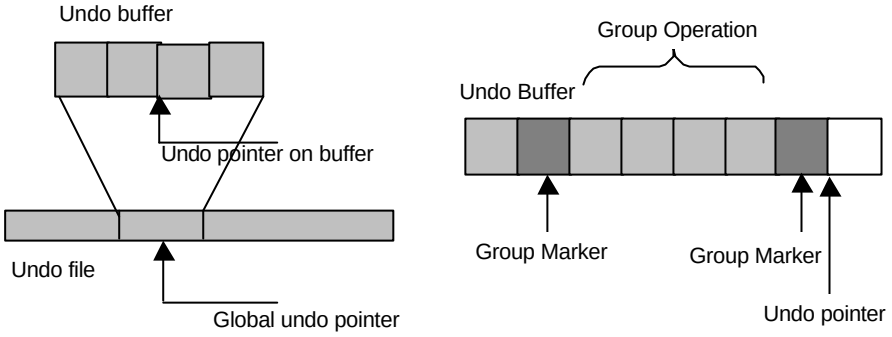

Figure 6. Undo buffers and undo operation for group of operations.

For group operations, like multiple object creation or deletion, we first put a special undo object called Group Marker to undo system, put all undo objects to system and to mark the end of the block. During the undo operation, if Undo manager finds a Group marker, it undoes all operations until it encounters another group marker (Figure 6.). The same mechanism holds for a redo operation.

General structure of the system is shown in (Figure 7), main components for providing flexibility in the CAD kernel are, project manager, tool manager, undo manager, object manager . A plug-in loader, layer system, snap and grid mechanisms, palet, font and symbol managers, and mathematical libraries for geometric operations are provided.

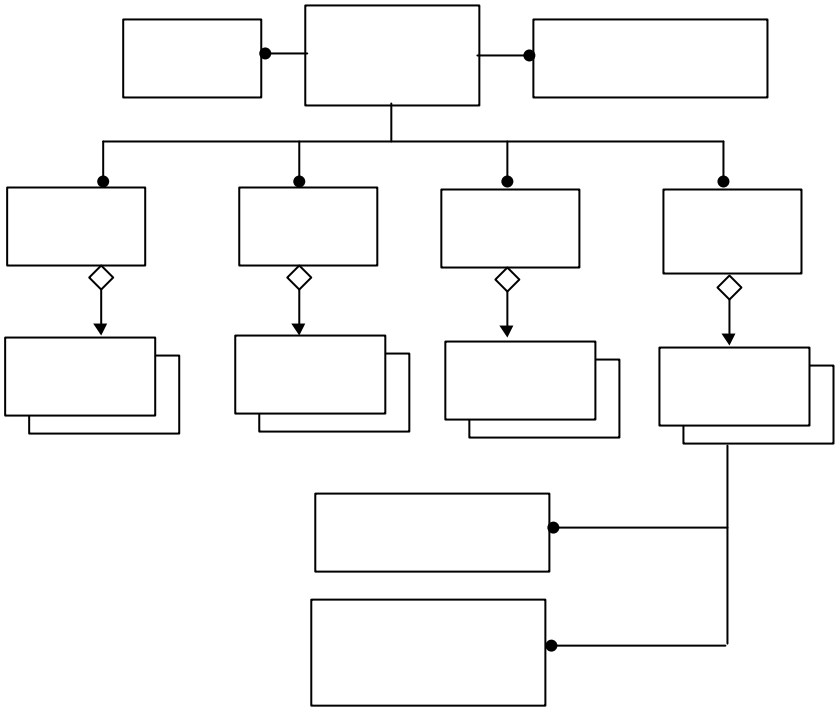

Figure 7. Block diagram for CAD kernel.

5. Conclusion

The use of object oriented design patterns can be very helpful in CAD like interactive systems. As every system has its specific needs and concerns, slight modifications on patterns or making compound design patterns may lead to better and more satisfying results. We have used several design patterns to guide us in the design of an interactive command system for a CAD software, with modifications at certain points to tailor them to the requirements of the system. We have observed that this approach has provided us flexible solutions to problems encountered during the development phase.

6. References

[1] Gamma E. Helm R., Johnson R., Vlissides J. "Design Patterns: Elements of Reusable Object Oriented Software", Addison Wesley, 1995.

[2] Vlissides J., "Tooled Compozite" C++ Tech. Report, September 1999, Vol. 11,No. 8.

[3] Bushmann F., Meunier R., Rohnert H., Sommerland P., Stal M. "A System of Patterns", John Wiley& Sons Ltd. ,1996.