SOFTWARE ARCHITECTURE IN OBJECT ORIENTED PROGRAMMING

Submitted by:
Oreoluwa Alebiosu
Kansas State University
Department of Computing and Information Sciences
Manhattan, KS 66506

May 10, 2013

ABSTRACT

Alebiosu, O. May 11, 2013. Software Architecture in Object Oriented Programming

Software development procedures have reached a defining stage where new developers aim to work on large software projects and experts aim to enhance their knowledge in software architecture. As freelance programming occupations have become more available to developers, the hopes of gaining an immediate knowledge in software development are common. This literature review addresses this issue towards software engineers and computer science majors, in order, to provide them with knowledge about object oriented programming, which is one of the most frequently used software development procedures in the technology industry. In this literature review, terms used in object oriented programming are defined, as well as, more complex concepts and ideas used in programming. Examples of object oriented programs are illustrated in this review. Diagrams showcased include class diagrams, object diagrams, real world implementation results, and software design patterns. Finally, this review summarizes the ideas and concepts then provides recommendation for further readings in the topic.

Key Terms: software architecture, class diagram, object diagram, software design patterns

Table of Contents

List of Figures

INTRODUCTION

This report is an effort to provide information for new developers on the basics of software architecture, focusing on Object Oriented Programming (OOP). More specifically, if the new developer has a minimum of three or more years of continuous development experience and an interest to learn more about recent techniques or practices in software architecture in object oriented programming.

Since the late 1980's, software architecture research has emerged as the principled study of the large-scale structures of software systems. From its roots in qualitative descriptions of empirically observed useful system organizations, software architecture has matured to encompass broad explorations of notations, tools, and analysis techniques. Whereas initially the research area interpreted software practice, it now offers concrete guidance for complex software design and development. It has made the transition from basic research to an essential element of software system design and construction (Eelses, 2006).

Software architecture describes a strategic design used in implementing component-based software systems by using blueprints and architectural design patterns (Schmidt, 2013). An example of an architectural design pattern in object oriented programming is Model-View-Controller. The Model-View-Controller (MVC) architectural pattern separates an application into three main components: the model, the view, and the controller. The model serves as the backend or database of the application, the controller performs all the necessary computation and enforces the user input and interaction between the model and view. Views are the components that act as the application's user interface. This architectural pattern is described in more detail in this report.

Object oriented programming is a slightly old design philosophy in software architecture and development; however, it is the most used design method in the technology industry. To understand the concept of object oriented programming, a few terms have to be defined. These terms are "class" and "objects". A class is a template for allocating a frame or namespace and defining functions (methods) that access the frame. An object is the frame or namespace that is allocated when you construct a new instance of a class template (Schmidt, 2013). As emphasis in programming were placed on objects and no longer actions or procedures, the adoption of object oriented programming came about as a programming language model organized around objects rather than actions, data, or logic.

Importance of Software Architecture

Architecture matters from a technical perspective, as it defines the structure of a software program, the behavior of each entity in the program, and allows for a more flexible adaptability of the program into a larger scale software program.

An Architecture Defines the Structure of Software Programs

The structure of different elements in a software system is easily identifiable if the software architecture of this program is designed.

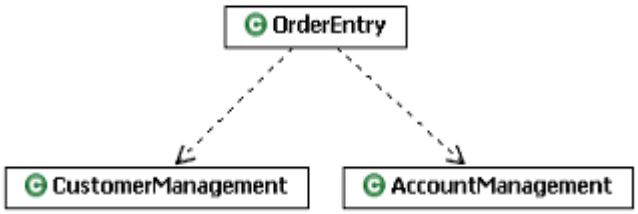

Figure 1: Unified Modeling Language (UML) class diagram showing structural elements, by Eelses (2006).

Source: Eeles, P. (2006). IBM - United States. *What is Software Architecture?* <http://www.ibm.com/developerworks/rational/library/feb06/eeles/> Accessed May 04, 2013.

The figure above shows a UML class diagram containing some structural elements that represent an order processing system. Here we see three classes -- OrderEntry, CustomerManagement, and AccountManagement. The OrderEntry class is depends on the CustomerManagement class and also the AccountManagement class. These different entities are easily identifiable and their dependencies are also illustrated in the figure above (Eeles, 2006).

An Architecture Defines Behavior of Different Objects

As well as defining structural elements, an architecture describes the interactions between these structural elements. Each class in software architecture has a defined attribute or capability, so when the class diagram from Figure 1 is executed, different interactions between the structural elements occur. From Figure 1 above, we may implement five different interactions. First, a Sales Clerk (user) creates an order using an instance of the OrderEntry class from Figure 1. The OrderEntry instance gets customer details using an instance of the CustomerManagement class. The OrderEntry instance then uses an instance of the AccountManagement class to create the order, populate the order with order items, and then place the order (Eeles, 2006). These interactions define the behavior of the objects in our software program, and even without running the program, we will understand this behavior by looking at the software architecture of the program.

Allows for Re-usability or Adaptability into a Different System

Software architecture constitutes a relatively small, intellectually graspable model for how a system is structured and how its elements work together, and this model is transferable across systems. In particular, it can be applied to other systems exhibiting similar quality attribute and functional requirements and can promote large scale re-use of the software as its different elements are easily identifiable (Verma, Dahiya, & Jain, 2010). In Kansas State University, K-State Online is as a simple, yet powerful, set of tools designed to deliver engaging and interactive online classroom ("KSOL index page", n.d.). K-State Online constitutes different objects that accomplish specified tasks. An example of these objects is a notepad that can be noticed on the index page of K-State Online's website. The notepad is an object in K-State Online and can be re-used in a different application, by obtaining the source code or executable program of the notepad and adding it as a plug-in into a different software system. The software architecture of a program makes it easy to know if the notepad depends on other objects. If the notepad depends on other objects, its dependencies as well shall be included as a plug-in into the different software system.

This report shall explain and describe different introductory concepts and ideas used in software architecture in object oriented programming by utilizing various diagrammatic representations such as class diagrams and object diagrams. The different components of a class diagram and object diagram are described in order to accomplish this task. The discussion section of this report illustrates various examples of class and object diagrams, and the relationship between different entities in the diagrams. Finally, a conclusion and recommendation is made for readers who are interested in more advanced topics in software architecture.

METHODS

Some of the diagrammatic representations of software architecture and development patterns in object oriented programs that are used in this review were developed on NClass and StarUML. Both of these development tools are free UML class designers used in building small-scale software architecture and design patterns. The class diagrams and object diagrams used in the discussion section to explain the relationships between different objects in object oriented programming were obtained from lecture notes provided to me by Professor Schmidt from the Department of Computing Science at Kansas State University. Definitions and descriptions of recent practices and techniques in object oriented programming were obtain from Microsoft's Developer Network library, IBM Developer work's library, and Association for Computing Machinery (ACM) Digital Library. These techniques are used in explaining the diagrammatic representations used.

RESULTS

Unified Modeling Language Diagrams

Unified Modeling Language, commonly known as UML, is an object-oriented system of notation that evolved from the work of Grady Booch, James Rumbaugh, Ivar Jacobson, and the Rational Software Corporation. These computer scientists aimed to standardize the pattern used to illustrate the relationship between objects in software architecture. Today, UML is accepted by the Object Management Group (OMG), an international, open membership, not-for-profit computer industry standards consortium, as the standard for modeling object oriented programs (*Software Design Tutorials,* n.d.). There are several types of UML diagrams; however, this report focuses only on the two major types which are class diagrams and object diagrams.

Class Diagrams

Class diagrams are used to describe the types of objects in a system and their relationships. Class diagrams model class structure and contents using design elements such as classes, packages and objects. It describes the information without reference to any particular implementation (Eelses, 2006). UML notations are used in describing the relationships between different entities in a class diagram. Below are UML notations that are mentioned on Microsoft Developer Network's library:

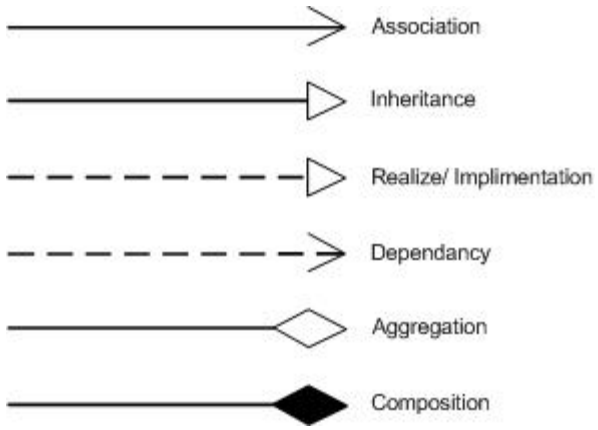

Figure 2: Common UML notations, by Nirosh (2011).

Source: Nirosh, L.W.C. (2011) *Introduction to Object Oriented Programming Concepts (OOP) and more.* < http://www.codeproject.com/Articles/22769/Introduction-to-prog Object-Oriented-Programming-Concept> Accessed April 12, 2013.

- Association: An association indicates that the system you are developing stores links of some kind between the instances of the associated types. It may also be defined as dependency (coupling): “A refers to B”, or “A needs B to compile correctly” and “A retains a handle to B in a field.”
- Inheritance: Also called generalization, means the specific classifier inherits part of its definition from the general classifier. The general classifier is at the arrow end of the connector. Attributes, associations, and operations are inherited by the specific classifier (Eelses, 2006).
- Realization: Realization indicates that a client model element is an implementation of the supplier model element, and the supplier model element is the specification.
- Dependency: This is an indication of a relationship that shows that an element, or set of elements, requires other model elements for their specification or implementation. The element is dependent upon the independent element, called the supplier (Eelses, 2006).
- Aggregation: An association representing a shared ownership relationship or an entity as a part of, or as subordinate to, another entity.
- Composition: An Association represents a structural relationship that connects two entities. This structural relationship stems from the attributes and variables of both entities.
    - Multiplicity (*): Multiplicity is indicated by an asterisk symbol (*). This symbol is used to define a range of number of instances or attributes of a class that is shared between the two entities that are connected by the arrow of composition (“UML Class Diagrams: Reference”, n.d.).

Object Diagrams

Objects act as an instance of a model element or entity in a class diagram. In software architecture, object diagrams provide a snapshot of the instances that are being executed in a system and the relationships between the instances (Eelses, 2006). Object diagrams allow developers to easily explore the behavior of a system at a point in time.

An object diagram is a UML structural diagram that shows the instances of the classifiers in models. Object diagrams use notation that is similar to that used in class diagrams. However, while class diagrams show the actual classifiers and their relationships in a system, object diagrams show specific instances of those classifiers and the links between those instances at a point in time (Eelses, 2006). Below is a discussion that describes how class diagrams and object diagrams are used in software architecture for object oriented programs.

DISCUSSION

Object diagrams can be illustrated in various formats depending on the developer. Below is an, easy to follow, diagrammatic representation of a typical object diagram.

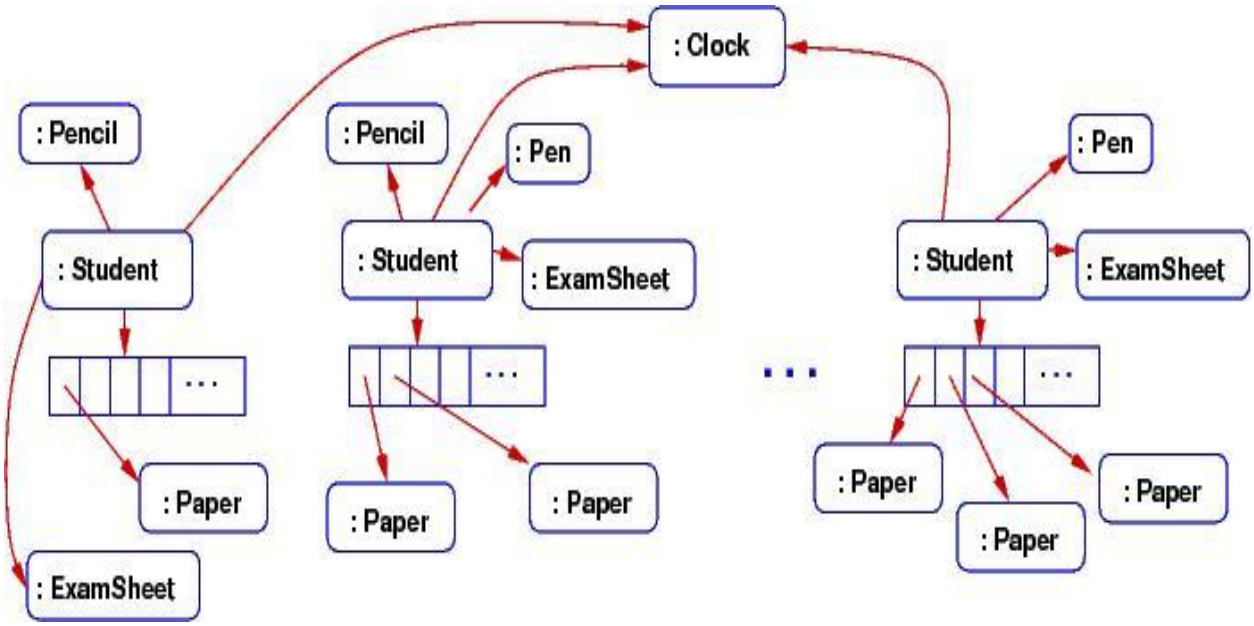

Figure 3: An object diagram of students taking an exam, by Schmidt (2013).

Source: Schmidt, D. (2013). CIS501 lectures: Software Architecture. <http://people.cis.ksu.edu/~schmidt/501s13/Lectures/Lecture03S.html> Accessed March 24, 2013.

Figure 1 illustrates the object diagram of what an exam simulation might look like while the software program is executing. The design of a computer simulation of students taking an exam will entail multiple students; each student owns a pencil, exam sheet, and papers (which is illustrated as a list in Figure 1). All students should have separate materials; however, they all depend on one clock which is noted by the proctor. The computer must construct objects for the entities just described.

The object diagram in Figure 1 gives big clues about what classes must be written for the simulation in the Figure to occur. Below is a class diagram that could generate the above object diagram from Figure 1:

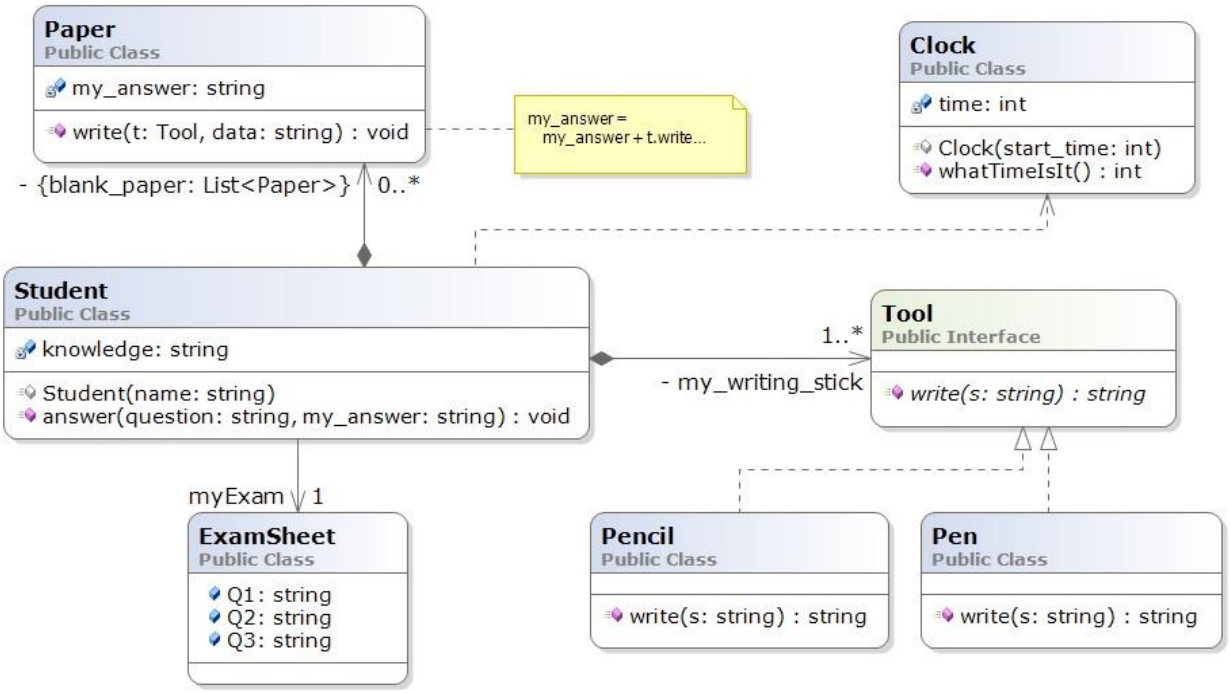

Figure 4: Class diagram that generates the object diagram of students taking an exam, by Schmidt (2013).

Source: Schmidt, D. (2013). CIS501 lectures: Software Architecture. <http://people.cis.ksu.edu/~schmidt/501s13/Lectures/Lecture03S.html> Accessed March 24, 2013.

Figure 2 is the class diagram that is designed to execute and create objects shown from Figure 1. The boxes represent entities such as classes, interfaces, and code fragments. Within a class box, the fields and methods are listed. Within an interface box, the methods are listed. The arrows are standard UML notations that define the relationships between each entity.

- The solid arrow (⟶) that goes from the Student class to Examsheet class is an association, which is also known as coupling. This means "a Student refers to and depends on an Examsheet", or "a Student retains a handle to an Examsheet in its field."
- The dashed arrow (---->) defines dependency, but where there is not a field to remember the entity depended upon.
- A solid arrow with a diamond at its base (◆⟶) defines composition. From the Figure 2 above, "a Student owns a Paper." This is a stronger form of dependency and occurs when the initial class constructs the composed entity.
- A dashed line with a white arrowhead (---|>) defines realization. Both Pencil and Pen class act as a Tool, therefore the Tool is an interface and its attributes are being realized by its sub-entities (Nirosh, 2011).

The word label on a solid arrow gives the name of the attribute that is declared within the class that depends on an entity, and the asterisk (*) information states the "multiplicity" which is how many of the objects is remembered. Names of collections are enclosed in braces, and the "-" means a certain variable or method is private to a particular class box. Figure 2 was able to effectively illustrate a small-scale software program that encompasses only 7 entities, with less than 4 attributes in its class box. However, a class diagram for a large-scale software program may contain hundreds, and possibly thousands, of entities that interact with each other. It is almost impossible to show a large-scale software architectural diagram with such simplicity by using a basic class diagram. This led to the introduction of the Model-View-Controller (MVC) pattern (Steve, 1992).

Model-View-Controller

The Model-View-Controller (MVC) pattern separates the modeling of the domain, the presentation, and the actions based on user input into three separate classes.

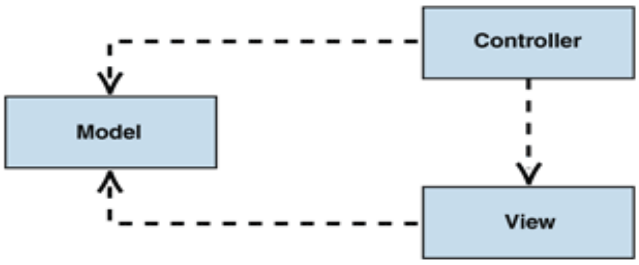

Figure 5: MVC pattern that depicts the structural relationship between the three objects ("Model View Controller", n.d.).

Source: Model View Controller. *Web Presentation Patterns.* <http://msdn.microsoft.com/en-us/library/ff649643.aspx> Accessed May 04, 2013.

Model: The model manages the behavior and data of the application domain, responds to requests for information about its state (usually from the view), and responds to instructions to change state (usually from the controller). The model in a software architecture pattern will contain all entities that a used as a backend or database in a software program (Schmidt, 2013).

View: The view manages the display of information. It shall contain all entities that are used interacting with the user.

Controller: The controller interprets the mouse and keyboard inputs from the user, informing the model and/or the view to change as appropriate. Controllers contain entities that are used to enforce the proper use of models and views in this design pattern ("Model View Controller", n.d.).

CONCLUSION

In summary, a class is a template for allocating a frame or namespace and defining functions (methods) that access the frame. An object is the frame or namespace that is allocated when you construct a new instance of a class template (Schmidt, 2013). Class diagrams and object diagrams are the two major types of Unified Modeling Language (UML) diagrams used in illustrating the interactions between different objects in object oriented programming. The knowledge of object oriented programs is necessary in large scale software development and this literature review has provided its readers with the most recent information and techniques in building class diagrams and object diagrams. It is important to understand the components that were used in designing the class diagrams illustrated, as they act as the major concepts and ideas that encompasses software architecture in object oriented programming. A further reading is recommended for this topic. The Model-View-Controller (MVC) software architecture was briefly discussed, and the scope of this design pattern and concept extends this literature review.

RECOMMENDATION

With the knowledge gained from this review, developers can approach the Model-View-Controller architecture and more complex design patterns which include factories, iterators, proxies, and wrappers. Below are suggested sources for further readings in this topic:

- A book written by Dr. Masaaki Mizuno, from the Department of Computing and Information Sciences, Kansas State University, called Software Architecture and Design Part 2: Object-Oriented Design Patterns.
- Microsoft Developer Network is also recommended for developers who are interested in obtaining code samples on how these patterns can be implemented and used in building different software programs.

REFERENCES

Eeles, P. (2006). IBM - United States. *What is Software Architecture?* Retrieved May 04, 2013, from http://www.ibm.com/developerworks/rational/library/feb06/eeles/

KSOL index page. *K-State Online.* Retrieved May 10, 2013, from http://public.online.ksu.edu/

UML Class Diagrams: Reference. *Reading Class Diagrams.* Retrieved May 04, 2013, from http://msdn.microsoft.com/en-us/library/vstudio/dd409437.aspx

Model View Controller. *Web Presentation Patterns.* Retrieved May 04, 2013, from http://msdn.microsoft.com/en-us/library/ff649643.aspx

Mizuno, M. (2013). *CIS 501 Software Architecture and Design (Part 2) – Design Patterns*.

Nirosh, L.W.C. (2011) *Introduction to Object Oriented Programming Concepts (OOP) and more.* Retrieved from http://www.codeproject.com/Articles/22769/Introduction-to-prog Object-Oriented-Programming-Concept.

Schmidt, D. (2013). *CIS501 lectures: Software Architecture*. Retrieved from http://people.cis.ksu.edu/~schmidt/501s13/Lectures/Lecture03S.html

Software Design Tutorials. Retrieved May 02, 2013, from http://www.smartdraw.com/resources/tutorials/uml-diagrams/

Steve, B. (1992). *Applications Programming in Smalltalk-80(TM): How to use Model-View-Controller (MVC).* Urbana: University of Illinois at Urbana-Champaign. Retrieved from http://st-www.cs.illinois.edu/users/smarch/st-docs/mvc.html

Verma, P. K., Dahiya, D., & Jain, P. (2010). Using Aspect Oriented Software Architecture for Enterprise Systems Development. *Digital Information Management (ICDIM), 2010 Fifth International Conference on*.