Software Architecture Styles a Survey

Ashish Kumar
Computer Science and Engineering,
Thapar University, Patiala -147004, Punjab, India

ABSTRACT

This research deals with an important issue in software development. It is concerned with software development with the help of software architecture styles. Software industry uses many architecture styles namely, pipe and filter, object oriented, event invocation, process control layered, data centered architecture etc. All these architecture styles have their own advantages and disadvantages as well. Therefore main objective of this research is to represent different architecture styles with their features and defects.

Keywords

Architecture styles, Advantages and disadvantages of different architecture styles, Software Architecture styles and patterns, Architecture styles in software engineering.

1. INTRODUCTION

An architecture style is a set of rules which tell how to develop a system. It tells how components are organized, how data is manipulated, how components communicate with each other, so on…..? Architecture is very important role player even for the making of a fountain pen to constructing a multistory building. Somehow if we are able to build our system once without architecture but it become deadly to make changes, future enhancement in the system without the proper architecture. A good architecture design has been a major factor in determining the success of a system.

Now a day's software architecture becomes the popular field for the software engineering researchers. Researches has been carried out in many different areas of architecture such as modules interfaces, software reuse, architecture design environment, domain specific environment etc.

The main purpose of software architecture is its benefits for both development and maintenance. For development point of view, it is very essential to be able to identify the common pattern, so that relationship among the system modules can be easily understood. In the complete life cycle of Software, maximum time spends during the maintenance e.g. understanding the code etc. If after development documentation is done properly; the effort for understanding the code can be reduced.

Using a wrong architecture can lead to disastrous result. A detailed understanding of software architecture is very

Helpful to analyze the complex system; it also gives a power to choose alternative architecture style. To get in depth understanding of importance of good architecture consider an example, suppose a small family wants to construct a three room set, it has two options either construct a very compact structure (focusing on the current needs only) or a well detailed design (focusing on the future needs too), which option would be better? Obviously smart answer will be second option, reason is clear family size may grow in future at that time modification feature should be possible in architecture, what if family decided to sell the house? A poor architecture style house is difficult to sell. In good architecture house, accessing and managing the things is easy. The same concept is also applied in software architecture i.e. our software system must be easy to use, reuse, easy to enhance etc.

Choosing the most appropriate architecture is not that much easy. It is impossible to answer all the questions arises in the Architect's mind during the software design like, when should a particular architecture should be chosen? What are the consequences of choosing one structural decomposition over another? Which architecture can be compared with others? And many more.

2. ARCHITECTURE STYLES

An architecture style is constrained by a particular principle of how to build a system and how components communicate to manipulate the data? E.g. sequential processing of data, hierarchy of components etc. Each architecture influence some quality attributes in a positive and some in a negative way, so each architecture style has its advantages and disadvantages.

In this research paper we mainly focus on the following architecture styles:

1. Pipe and filter architecture
2. Object oriented architecture
3. Layered architecture
4. Data- centered architecture
5. Interpreter
6. Event based implicit invocation
7. Process control

2.1 Pipe and Filter architecture

This architecture contains set of components and connectors. Components are computations unit also known as filters. These filters are connected with the help of connectors (known as pipe) through which data flow from one filter to other. Each filter take data as input then perform some computation on it and send it to the other filters for further processing. Interestingly filters do not know the identity of other filters either they are connected or not and all the filters are independent entities. These filters can be used in sequential or in parallel or in both.

Pipe and filter style has mainly three specialization include *pipelines* which allow only linear sequence of filters; *bounded pipes* which tells about amount of data that can reside on a pipe; *typed pipes* which allow only a specific type of data to be passed between the filters.

Well known example of pipe and filter is compiler. A compiler has many filters (lexical analyses, parsing, sematic and code generation) through which our program passes and we get final machine code after this. Other well-known examples of pipe and filter style are programming in Unix shell [1], signal processing domain [2], parallel programming [3], functional programming [4] and distributed systems.

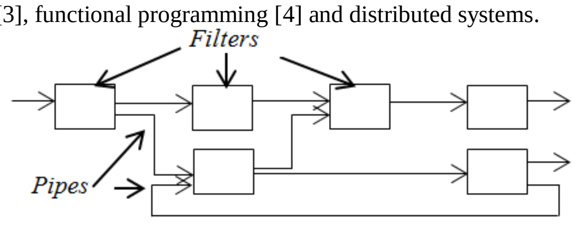

Fig.1 Pipes and filters.

Advantages of pipe and filter architecture are as follows:

1. Easy to understand the overall input and output behavior of the system.
2. Support reuse.
3. Support deadlock analyses feature.
4. Concurrent execution which improve performance.
5. Easy to maintain and enhance; new filter can be added or can replace older filters with improved ones.

Disadvantages of pipe and filter architecture are as follows:

1. Batch oriented processing.
2. Not suitable for interactive applications
3. Must agree on lowest common denominator on data transmission which leads to loss of performance and increase complexity.

2.2 Object oriented architecture

In this style, data representation and their associated primitive operations are encapsulated in an abstract data type or object. Instances or objects of abstract data type are the components of this style. Objects interact with each other with the help of functions and procedure call. A problem can be decomposed in to set of small problems; solutions of these small problems are shared with other small problems with the help of procedure invocation to solve a big problem. It can be multithreaded or single threaded.

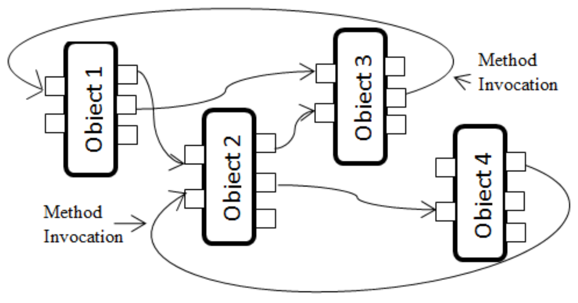

Fig. 2: Object Oriented Architecture.

Advantages of object oriented architecture are as follows:

1. Data hiding i.e. changes in an object is possible without affecting its clients.
2. Allow designer to decompose problem in to collection of interacting objects.

Disadvantages of object oriented architecture are as follows:

1. Object must know the identity of other objects with which it want to interact via procedure call.
2. Whenever identity of object changes it must modify all other objects that explicitly invokes it.
3. Side effect problem[5]

Object oriented architecture has mainly two variants- *client server* and *object broker*. In client server, objects (clients) do not need to know anything about each other. Clients just need to know about the server with which it has to be connected and further responsibility is of server only.

In object oriented, a broker is used between clients and servers. Now clients no longer need to know the server they are using. Clients only need to know the broker and broker decide with which server it has to be connected. There can be many brokers and many servers. But it has serious drawback that at peak time broker can be bottleneck and performance of system degraded.

2.3 Layered architecture

It is a hierarchical organization, each layer provide services as a server to the layer above it and serving as a client the layer below. Depending on the number of layers architecture can be 2-tier architecture (only 2 layers present), 3-tier architecture (only 3 layers present) and so on. Each layer has its own responsibilities. Together all the layers work to achieve a single goal. Two well-known examples of layered architecture are *operating system* and *OSI reference model*.

In operating system, mainly three types of layers are used- kernel layer (closer to hardware), it provides the services to the utilities (compilers, drivers etc.) to be installed and then application layer come in to the existence which help user to interact with the system.

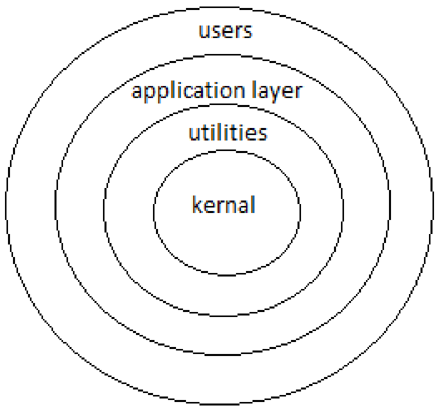

Fig. 3: Different layers of operating system.

In OSI reference model, seven layers are used applicatic layer, presentation layer, session layer, transport laye network layer, data link layer and physical layer (in the order upper to lower layer). Physical layer provide its services to the data link layer and data link layer provide its services to the network layer and so on.

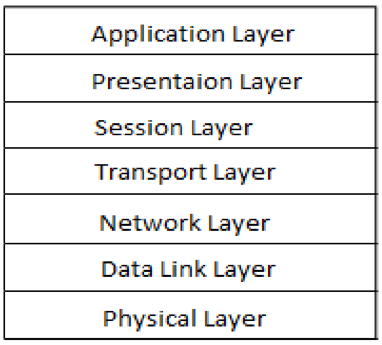

Fig.4: Different layers of OSI reference model.

Layered architecture has mainly two variants- *open layered architecture* and *closed layered architecture*. In open layered architecture an upper layer can use the services of any below layer directly. As a result it increases the dependency between the layers, so layers are not reusable.

In closed layered architecture, an upper layer can use the service of layer directly below it. In this architecture layers are less dependent on each other hence reduce the impact of changes and increase reusability. But its performance is poorer then the open layered architecture as it has to traverse all the layers for performing any computation.

Advantages of layered architecture are as follows:

1. High abstraction level.
2. Easy to enhance (new layers can be easily added or deleted).
3. Reusable (change in one layer can affect only two adjacent layer i.e. one upper layer and one lower layer).

Disadvantages of layered architecture are as follows:

1. Level of abstraction is difficult to decide.
2. Low performance (layer by layer path need to be followed).
3. Dividing every system into layers and deciding the functionality of layers is difficult.

2.4 Data-centered architecture

As its name implies our main focus is on data. Several knowledge experts use it to get a common conclusion. These knowledge experts access the common data in synchronized manner to preserve the consistency of data. Transactions on the data are in same order for all the knowledge sources. In this architecture our focus is mainly on data that is why it is called data centered architecture.

Blackboard architecture has mainly three components- *knowledge sources (KS), blackboard data structure* and *controller*.

- **The knowledge source (KS):** these are independent knowledge specialists. Each provides specific expertise needed for the application. These experts interact with each other throughout the life of problem to achieve a common target.
- **The blackboard data structure:** is a shared repository of problems, partial solutions, contributed information and suggestions. Knowledge sources make changes to the data repository that lead incrementally to a solution to the problem. It can be treated as dynamic library of contributions to the current problem that have been recently published by other knowledge sources.
- **The controller:** it controls overall flow of problem solving activities in the system. It is used mainly to organize the knowledge sources so that they can make use of data repository in a coherent and effective manner.

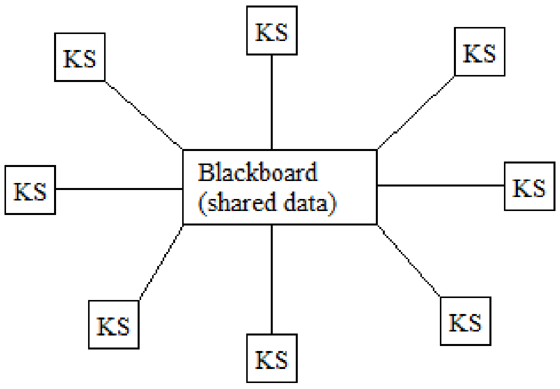

Fig. 5: The blackboard architecture.

To make these components more understandable take an example, in a class teacher write a problem on the board and ask students to come in front and solve it one by one. Each student write the solution according to his thinking then next student came and can modify or suggest any alternative for providing better way to solve the problem. This process is keep repeating till we get a good optimized solution. Here students act as knowledge experts, board which share problem, intermediate and final solution act as blackboard data repository. And teacher which control all the activities of the process i.e. order of accessing data repository by a particular student, state of data repository etc. act as controller.

Advantages of blackboard architecture are as follows:

1. Suitable for complex problem e.g. speech and pattern recognition.
2. Very helpful for network based applications.
3. Useful in the systems that involve shared access to data with loosely coupled agents [6].

Disadvantages of blackboard architecture are as follows:

1. Not suitable for every problem.
2. Designing of good controller is very important.
3. If number of knowledge sources requested to access the shared data increases then bottleneck problem arises at the controller.

2.5 Interpreter

An interpreter basically used to produce a virtual machine. It is suitable for applications in which most appropriate machine is not directly available. Java language uses interpreter. When a java program is compiled with the help of JVM (Java virtual machine) JVM create a byte code which is an intermediate code not binary code. Now this byte code can be run on any machine. It has four states- one for engine and rest three are memories. These states are:

1. Interpreter engine for executing the program.
2. Current state of interpreter engine.
3. Program being interpreted.
4. Current state of program being interpreter.

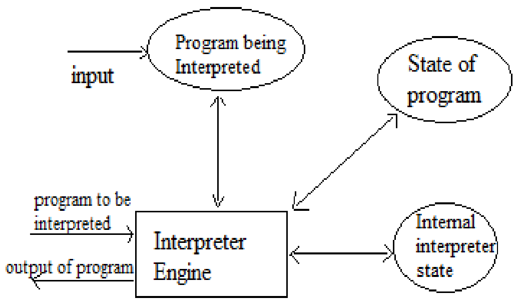
input
Program being Interpreted
State of program
program to be interpreted
output of program
Interpreter Engine
Internal interpreter state

Fig. 6 Interpreter.

Advantages of interpreter are as follows:

1. It produces the machine independent code.

Disadvantages of interpreter are as follows:

1. Slow performance.

2.6 Event based implicit invocation

In event based implicit invocation instead of invoking a procedure directly, a module or a component broadcast one or more events. While other components can register an interest in an event by associating a procedure with it. Whenever the event is announced, the system invokes implicitly all the procedures that have been registered their interests for the event.

Interface of component in an implicit invocation style provides both a collection of procedures and set of events. A component can register some of its procedures with events of the system. This will cause procedures to be invoked when their associated events are announced at run time.

Interestingly the announcer of events even do not know which component will be affect by which event. Hence even components cannot make assumptions about what processing will happen as a result of their events. For web application developed in programming language like in java also announces event when user interact with application and then corresponding listener handles that announcement for example when user clicked then corresponding listener mouseClicked listen this event and handle it.

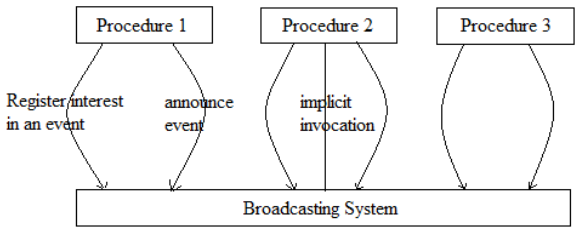

Fig. 7 Implicit invocation-event handling

This architecture is used in many applications for example in database systems to ensure consistency constraints[7], in user interfaces to separate presentation of data from applications that manage the data[8], by syntax-directed editors to support incremental checking[9].

Advantages of implicit invocation are as follows:

1. It supports reuse.
2. System evolution is easy in this architecture.
3. It is suitable for asynchronous communication.

Disadvantages of implicit invocation are as follows:

1. Components do not have control over computation since they can generate events only.
2. Responses to the events are not ordered.
3. Analysis via pre and post condition is difficult.
4. Exchange of data requires global data or shared repository, so resource management become challenging.

2.7 Process control

This architecture is not widely used in the software industry. In software design other architecture styles are dominated over this architecture. This architecture is characterized by the components and their relationship.

A continuous process converts input material in to product with specified properties by performing operations on the inputs and intermediate products. Process control architecture involves several paradigms, these are as follows:

- **Input variable** measures the input to the system.
- **Controlled variable** are those variables whose value the system is intend to control.
- **Manipulated variable** are those variables which can be changed by the control system in order to monitor the process.
- **Set point** in order to maintain specific property of output of process, we set specific standards or values, these standards are called set points.
- **Open loop system** if a process can run without any surveillance, that system is called open loop system. E.g. a hot air furnace system that uses a constant burner setting to raise the temperature of air that passes through. If we use timer to turn on or off at regular interval even then it is called open loop system.

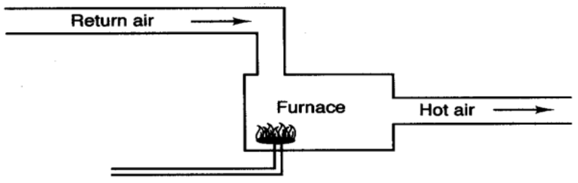

Fig.8 Open loop system

- **Close loop system** in real world rarely it happens, we do not need to control the physical processes. Many times property like temperature, pressure and flow rate are need to be monitor and their values are used to change the setting of apparatus like valves, heaters etc. such systems are called close loop system.

This architecture is used in industrial production lines, power stations, chemical engineering etc.

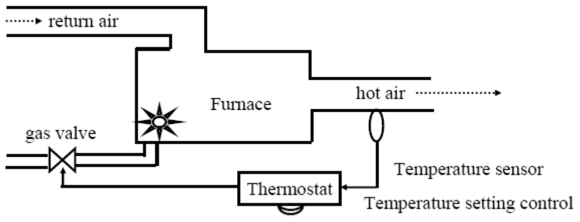

Fig.9 Close loop system

Advantages of process control are as follows:

1. Suitable for the control of the processes even at runtime especially where control algorithm is subjected to change.

2. Suitable when software is embedded in a physical system that involves continuing behavior.

Disadvantages of process control are as follows:

1. Difficult to specify the timing characteristics and responses to disturbances.

3. CONCLUSION

Each architecture style has its own merits and demerits. It may happen that one architecture may fit for some application but not fit to other application. None of the architecture is suitable for all the application but it depends on the architect which architecture he prefer for what type of application depending upon the need of the application.

Table 1. Architecture styles and their qualities

Quality → Arch. Style ↓	Performance	Scalable	Simplicity	Reusable
Pipe and Filter	+-	++	++	++
Object Oriented	+	+-	-	+
Layered Arch.	-	+-	+	++
Data Centered Arch.	+	+	--	+
Interpreter	--	+	+	+
Implicit Invocation	+	--	+-	+
Process Control	+	+-	-	+

4. ACKNOWLEDGMENT

I would like to express the deepest appreciation to Dr. Parteek Bhatia, Assistant Professor at Thapar University, Patiala who has attitude and substance of a genius. Without his expert guidance and persistent help this paper would not have been possible in this shape.

5. REFERENCES

[1] Maurice J. Bach. The design of the UNIX operating system, chap. 5,pp. 111-119. Software series. Prentice Hall, 1986.

[2] Norman Delisle and David Garlan. Applying formal specification to industrial problems: A specification of an oscilloscope. IEEE software, 7(5):29-37, September 1990.

[3] J.C. Browne, M. Azam, and S. Sobek. Code: A unified approach to parallel programming. IEEE Software, July 1989.

[4] G. Kahn. The semantics of a simple language for parallel programming. Information Processing, 1974.

[5] Mary Shaw and David Garlan, Software Architecture, Prentice Hall of India, 2004.

[6] Vincenzo Ambriola, Paolo Ciancarini, and Carlo Montangero. Software process entactment in Oikos. In proceedings of the Fourth ACM SIGSOFT Symposium on Software Development Environments, SIGSOFT Software Engineering Notes, 183-192. Irvine, California, December 1990.

[7] Carl Hewitt. Planner: A language for proving theorems in robots. In proceedings of the First international Joint Conference in Artificial Intelligence, 1969.

[8] G.E. Krasner and S.T.Pope. A cookbook for using the model-view-controller user interface paradigm in smalltalk-80. Journal of object oriented programming, 1(3):26-49, August/September 1988.

[9] A.Nico Habermann and david S.Notkin. Gandalf: Software development environment. IEEE Transactions on Software Engineering, SE-12(12):1117-1127, December 1986.