Software Engineering

Lecture 8

Architectural Design

Dr. Obuhuma James

Description

This topic starts up the design process by exploring architectural design as a way of describing a top-level structure of a software, including its components. The various types of software architectures will be covered, each with its associated pattern. Quality attributes of software architectures will also be outlined. Finally, the benefits and limitations of architectural design will be brought out.

Learning Outcomes

By the end of this topic, you will be able to:

- Describe architectural design as applied in Software Engineering.
- Differentiate among the various types of architectures with their associated patterns.
- Develop an architectural design for any given set of system requirements.

Overview of Architectural Design

The last two topics explored system modeling as a way of transforming user requirements into models. The development of system models now opens an avenue for generation of an architectural design. Architectural design is the design process of identifying sub-systems making up a system and the framework for sub-system control and communication [1]. The end product of the architectural design process is a software architecture.

Architectural Design

Architectural design can be perceived to be an early stage of the system design process, representing a link between requirements specification and design processes [1]. It is more of a representation of a system aimed at bringing out a better understanding of the anticipated system behaviour. Architectural design entails identification of major system components, their properties, and the communication among them. Figure 1 shows an example of an architecture of a packing robot control system where the major components are a vision system, object identification system, arm controller, gripper controller, packaging selection system, packaging system, and a conveyor controller. Links among the components are shown by the lines with arrowheads.

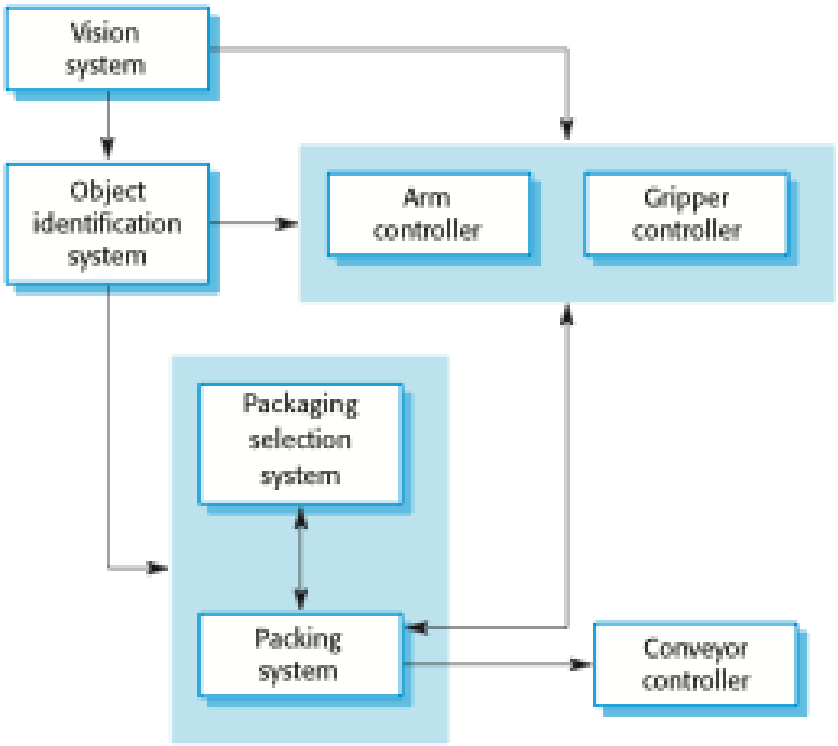

Figure 1. Example Architecture of a Packing Robot Control System

The use of simple, informal block diagrams showing entities and their relationships is the most frequently used approach for developing architectural designs [1]. This is normally achieved using box and line diagrams, which come as abstract models. According to Sommerville [1], the approach is heavily criticized for lack of semantics and not being able to show either relationship types between entities or the visible properties of entities in the architecture. They are however still sufficient for communication with stakeholders and project planning. Thus, the determinant factor on the required semantics is the intended use of the architectural model.

A system's architecture can be classified broadly as architecture in the small and architecture in the large.

- Architecture in the small – focuses on the architecture of individual programs with respect to the manner in which they are decomposed into components.
- Architecture in the large – focuses on the architecture of complex enterprise systems distributed across various computers with different owners and managers. Such enterprise systems include other systems, programs, and program components.

In summary, architectural designs serve as blueprints for both the system and the project by providing an abstraction for management of system complexity and establishment of

communication and coordination mechanism among system components. Some of the components of software architectures include business strategy, quality attributes, human dynamics, design, and an IT environment.

According to Sommerville [1], architectural design is a creative process that varies depending on the type of system under development. However, several common design decisions exist that may affect the nonfunctional characteristics of a system. These include [1]:

1. Is any generic application architecture available for use?
2. How will the system be distributed?
3. What architectural styles are appropriate?
4. How will the system be structured?
5. How will the system be decomposed into modules?
6. What control strategy should be used?
7. How will the architectural design be evaluated?
8. How should the architecture be documented?

Architectural designs heavily support reuse since systems in similar domains often portray similar architectures embracing standard domain concepts [1]. Thus, generally, application product lines are founded and built around a core architecture with just a few variations inclined to customer unique requirements.

Software Architecture Types and Patterns

There exist different software architecture types with their associated patterns. An architectural pattern in this case refers to a general, reusable solution to a commonly occurring problem in a software architecture within a given context. According to Sommerville [1], architectural patterns are a means of representing, sharing and reusing knowledge. They are stylized descriptions of good design practice, that has been tried and tested in different environments. Patterns should include information about when they are and when they are not useful. They may be represented using tabular and graphical descriptions.

1. Layered architecture

The layered architecture takes a pattern that defines different levels of abstraction, referred to as layers, with each layer providing services to the next higher layer. These layers may include the presentation layer, application layer, business logic layer, and the data access layer. This hence means that it can be used to structure programs that can be decomposed into groups of subtasks, each of which takes up its own layer. Figure 2 shows a generic layered architecture. Such an architecture is best suited for general desktop applications and web-based applications.

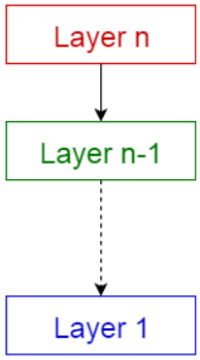

Figure 2. Layered Architecture Pattern

2. Client-server architecture

The client-server architecture takes a pattern composed of two entities, namely, client and server, as shown in Figure 3, such that a server entity acts as a service provider to multiple client entities. The server is setup is such a manner that allows for continuous listening to client requests and providing the requested services to clients. The architecture is best suited for online web-based applications.

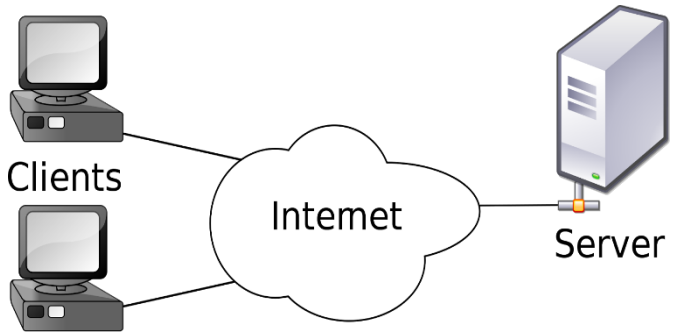

Figure 3. Client-server Architecture Pattern

3. Master-slave architecture

The master-slave architecture takes a pattern that almost relates to the client-server architecture pattern. It is composed of two entities, namely, master and slave, as shown in Figure 4, such that a master entity shares tasks among slave entities. It then aggregates results returned from slaves into the final result for the main task. The architecture is best suited for database replication applications and the master-slave drives concept used in computing systems.

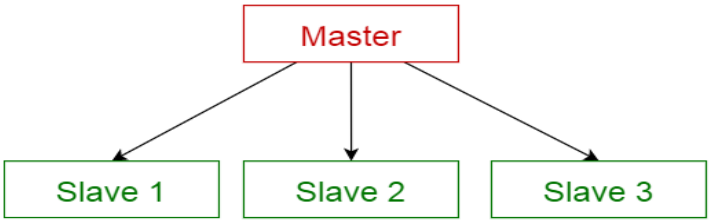

Figure 4. Master-slave Architecture Pattern

4. Pipe-filter architecture

The pipe-filter architecture takes a pipeline kind of pattern used to structure systems that produce and process a stream of data. Check points are positioned within the processing pipe, as shown in Figure 5, such that each processing step has a filter component before proceeding to the next step, making the architecture support buffering or synchronization processes. The architecture is best suited for applications like compilers which performs checks entailing lexical analysis, parsing, semantic analysis, and code generation.

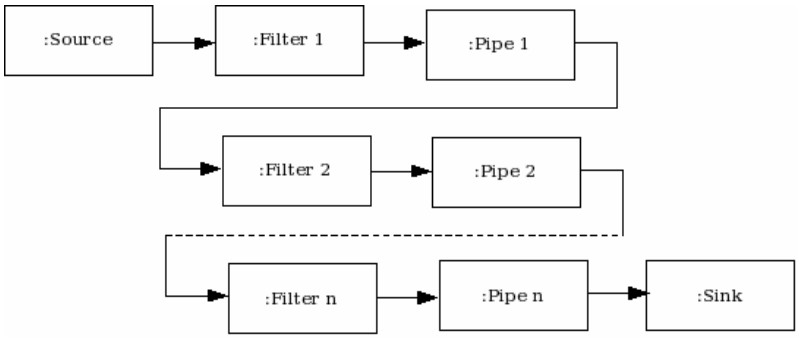

Figure 5. Pipe-filter Architecture Pattern

5. Peer-to-peer architecture

The peer-to-peer architecture takes a pattern composed of many entities referred to as peers. A peer in this case may function both as a client or a server, as shown in Figure 6. Thus, it may at some point be the one requesting for a service from other peers, while at other points in time, it may be the service provider for other peers. The dynamic change of roles for any given entity is what differentiates the architecture from the concept used by the client-server architecture and the master-slave architecture. The architecture is best suited for file sharing networks, multimedia protocols, and cryptocurrency-based systems like Bitcoin and Blockchain.

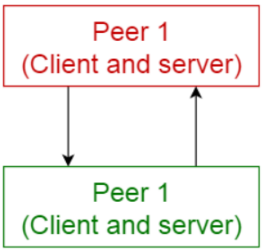

Figure 6. Peer-to-peer Architecture Pattern

6. Model-view-controller architecture

The model-view-controller architecture takes a pattern that divides an interactive application into three parts, namely, the model, view, and controller. The model part takes care of the core functionality and data; one or many views are used to display information to the user; while the controller handles user input. The model-view-controller pattern is also known as MVC pattern. The architecture is best suited architectures for World Wide Web applications in major programming languages and for web frameworks.

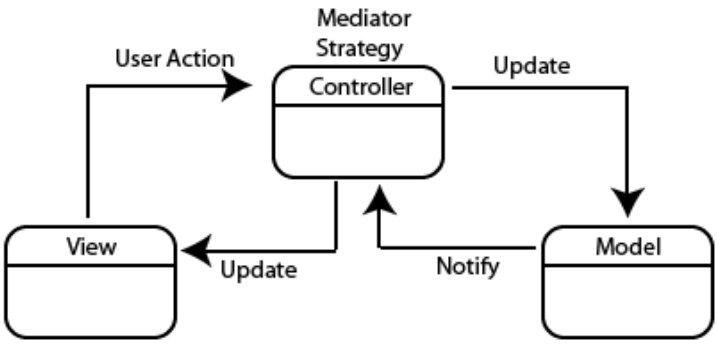

Figure 7. Model-view-controller Architecture Pattern

Other architectures worth further reading includes Broker architecture, Event-bus architecture, Blackboard architecture, and the Interpreter architecture.

Table 1 outlines a comparative analysis of the architectures in terms of their advantages and disadvantages.

Table 1. Comparison of Architecture Patterns

Name	Advantages	Disadvantages
Layered	A lower layer can be used by different higher layers. Layers make standardization easier as we can clearly define levels. Changes can be made within the layer without affecting other layers.	Not universally applicable. Certain layers may have to be skipped in certain situations.
Client-server	Good to model a set of services where clients can request them.	Requests are typically handled in separate threads on the server. Inter-process communication causes overhead as different clients have different representations.
Master-slave	Accuracy - The execution of a service is delegated to different slaves, with different implementations.	The slaves are isolated: there is no shared state. The latency in the master-slave communication can be an issue, for instance in real-time systems. This pattern can only be applied to a problem that can be decomposed.
Pipe-filter	Exhibits concurrent processing. When input and output consist of streams, and filters start computing when they receive data. Easy to add filters. The system can be extended easily. Filters are reusable. Can build different pipelines by recombining a given set of filters	Efficiency is limited by the slowest filter process. Data-transformation overhead when moving from one filter to another.
Broker	Allows dynamic change, addition, deletion and relocation of objects, and it makes distribution transparent to the developer.	Requires standardization of service descriptions.
Peer-to-peer	Supports decentralized computing. Highly robust in the failure of any given node. Highly scalable in terms of resources and computing power.	There is no guarantee about quality of service, as nodes cooperate voluntarily. Security is difficult to be guaranteed. Performance depends on the number of nodes.
Event-bus	New publishers, subscribers and connections can be added easily. Effective for highly distributed applications.	Scalability may be a problem, as all messages travel through the same event bus
Model-view-controller	Makes it easy to have multiple views of the same model, which can be connected and disconnected at run-time.	Increases complexity. May lead to many unnecessary updates for user actions.
Blackboard	Easy to add new applications. Extending the structure of the data space is easy.	Modifying the structure of the data space is hard, as all applications are affected. May need synchronization and access control.
Interpreter	Highly dynamic behavior is possible. Good for end user programmability. Enhances flexibility, because replacing an interpreted program is easy.	Because an interpreted language is generally slower than a compiled one, performance may be an issue.

Quality Attributes of Software Architectures

To achieve better architectural design, the following quality attributes should be put into consideration:

1. Design qualities – entails a consideration of conceptual integrity, maintainability, and reusability.
2. Run-time qualities – entails a consideration of interoperability, manageability, reliability, scalability, security, performance, and availability.
3. System qualities – entails a consideration of supportability, and testability.
4. User quality – entails a consideration of usability.
5. Architecture quality – entails a consideration of correctness.
6. Non-runtime qualities – entails a consideration of portability, integrity, and modifiability.

7. Business qualities – entails a consideration of cost and schedule, and marketability.

According to Sommerville [1], the following are examples of aspects to consider for some critical attributes under the design and run-time qualities:

- Performance – critical operations should be localized with communications minimised. The best approach is the use of large rather than fine-grain components.
- Security – for better security, a layered architecture with critical assets in the inner layers should be used.
- Safety – safety-critical features should be localized in a small number of sub-systems.
- Availability – redundant components and mechanisms for fault tolerance should be considered for inclusion.
- Maintainability – the use of fine-grain, replaceable components is encouraged.

Benefits of Architectural Design

1. Architectures facilitate stakeholder communication since they are used as a discussion focus by system stakeholders.
2. Used in system analysis as a way of ensuring that the system meets its nonfunctional requirements to a large extend.
3. Architectures can be used for large-scale reuse which saves on resources including time and money.

Thus, architectural designs provide a way of facilitating discussion about the system design and also act as documentation for the system being designed.

Limitations of Architectural Design

Despite the advantages of architectural design stated in the previous section, the following limitations could associate with architectural design

1. There are no specific tools and standard ways of representing architectures.
2. It may not be clear on analysis methods for predicting whether a given architecture will lead to an implementation that fully addresses requirements.
3. The development team may lack awareness on the importance of architectural design.
4. Lack of awareness on software architect's role and poor stakeholders' communication.

5. Lack of knowledge on the design process, experience, and evaluation.

Summary

The topic has explored architectural design at length. A comparative discussion and analysis of the different types of software architectures and their associated patterns. Furthermore, quality attributes of software architectures have been mentioned as a guide for architects. The topic concludes with an outline of the benefits and limitations of architectural design.

Check Points

1. Discuss architectural design as used in Software Engineering.
2. Explain how architectural design embraces software reuse.
3. Compare and contrast the various types of architectures and patterns.
4. Describe the quality attributes that guide in software architectural design.
5. State and explain the benefits of architectural design.
6. State and explain the limitations of architectural design.

Learning Resources

Core Textbooks

1. Sommerville, I., Software Engineering, 10th Edition, Addison Wesley, 2016.

Other Resources

2. Pressman, R. S. (2005). Software engineering: a practitioner's approach. Palgrave macmillan.

References

[1] Sommerville, I., Software Engineering, 10th Edition, Addison Wesley, 2016.

[2] Pries, K. H., & Quigley, J. M. (2010). Scrum project management. CRC press.

[3] Schwaber, K. (2004). Agile project management with Scrum. Microsoft press.

[4] Gotterbarn, D., Miller, K., & Rogerson, S. (1997). Software engineering code of ethics. Communications of the ACM, 40(11), 110-118.

[5] Pressman, R. S. (2005). Software engineering: a practitioner's approach. Palgrave macmillan.

[6] Kendall, K. E. and Kendall, J. E., Systems Analysis and Design, 8th Edition, Pearson, 2011.