Hyperiondev

TASK

System Architecture

Visit our website

Introduction

Welcome to the System Architecture Task!

Architectural design is concerned with understanding how a system should be organised and designing the overall structure of that system. In this task, you will be introduced to system architecture and architectural design.

Get in touch

Connect for support

Remember that with our courses, you're not alone! You can contact your mentor to get support on any aspect of your course.

The best way to get help is to login to **www.hyperiondev.com/portal** to start a chat with your mentor. You can also schedule a call or get support via email.

Your mentor is happy to offer you support that is tailored to your individual career or education needs. Do not hesitate to ask a question or for additional support!

What is System Architecture?

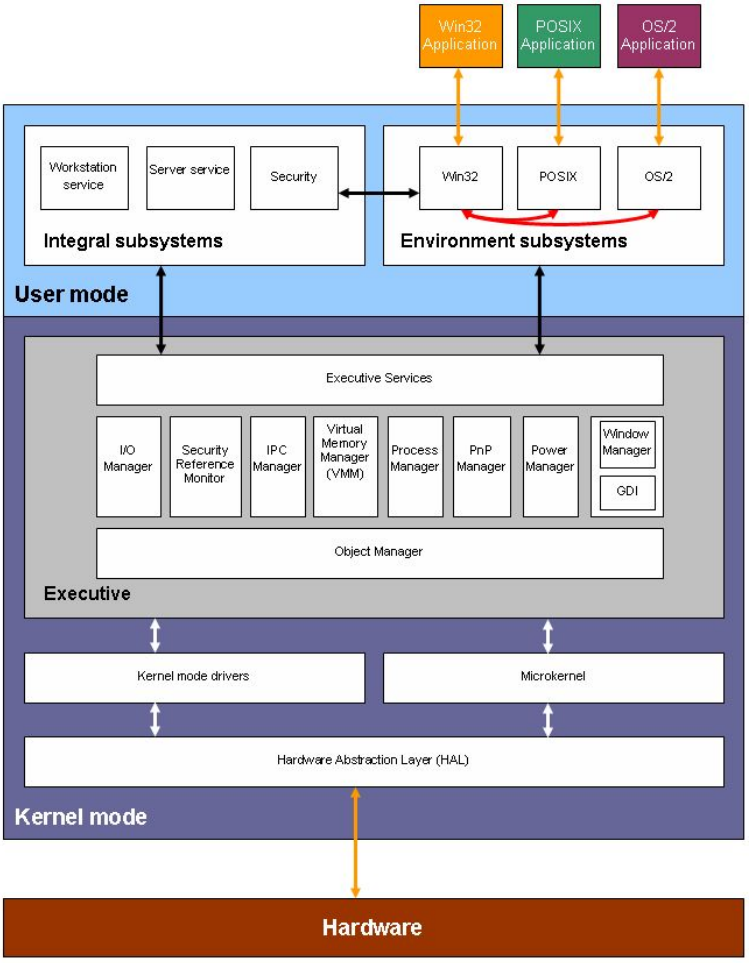
According to Philippe Kruchten, Grady Booch, Kurt Bittner, and Rich Reitman, who derived and refined a definition of architecture:

"Software architecture encompasses the set of significant decisions about the organization of a software system including the selection of the structural elements and their interfaces by which the system is composed; behavior as specified in collaboration among those elements; composition of these structural and behavioral elements into larger subsystems; and an architectural style that guides this organization. Software architecture also involves functionality, usability, resilience, performance, reuse, comprehensibility, economic and technology constraints, tradeoffs and aesthetic concerns."

As you can see the definition focuses on system structure. The structure of a system can simply be defined as the mechanisms that the system employs to solve the common problems faced by a system. An architecture is created to describe the structure of the system to be built and how that structure supports the requirements.

Architectural design is the first stage in the software design process and it links together both design and requirements engineering. Architectural design identifies the main structural components in a system and the relationships between them. There is quite a big overlap between requirements engineering and architectural design. A system specification should ideally not include any design information, however, this is unrealistic. Architectural decomposition is used to structure and organise the system specification. As part of the requirements engineering process, you might propose an abstract system architecture where you associate groups of system functions or features with certain large-scale components or subsystems. You can then use this decomposition to discuss the requirements and features of the system with stakeholders.

System architectures can be modeled using simple block diagrams like the one given below. Each box represents a component and boxes within other boxes indicate that the component has been decomposed to sub-components. Arrows indicate the direction of data or control signals which are passed from component to component. These block diagrams show a high-level picture of the system structure. This allows many different types of people from many different backgrounds to easily understand the system structure. However, these diagrams do have their drawbacks as they do not show the type of the relationships among system components or the components' properties that are externally visible.



Block Diagram Showing The Microsoft Windows 2000 Operating System Architecture (wikipedia.org)

Block diagrams are a good way of describing the system architecture during the design process, as they aid in supporting communications between the various people involved in the process. However, if the architecture of a system is to be thoroughly documented then it is better to use a notation with well-defined semantics for architectural description.

Why is System Architecture Important?

Software architecture affects the performance, robustness, distributability, and maintainability of a system. There are three advantages of explicitly designing and documenting software architecture:

1. **Stakeholder communication:** The architecture is a high-level presentation of the system that may be used as a focus for discussion by many different stakeholders.

2. **System analysis:** Making the system architecture explicit at an early stage in the system development requires some sort of analysis. Architectural design decisions have a great effect on whether or not the system can meet critical requirements such as performance, reliability, and maintainability.

3. **Large-scale reuse:** A model of a system architecture is a compact, manageable description of how a system is organised and how the components work together. For systems with similar requirements, the system architecture is often the same and so can support large-scale software reuse.

Software architecture can serve as a design plan for the negotiation of system requirements, and as a means of structuring discussions with various stakeholders. It is also an extremely important tool for complexity management as it hides details and allows the designers to focus on the key system abstractions.

Architectural Design Decisions

With architectural design, you design a system organisation that will satisfy the functional and nonfunctional requirements of a system. Architectural design is a creative process and therefore the activities within the process depend on the type of system being developed, the background and experience of the system architect, and the specific requirements for the system. You should think of architectural design therefore as a series of decisions to be made and not a sequence of activities.

The particular architectural style and structure that you choose for a system should depend on the nonfunctional system requirements because of the close relationship between nonfunctional requirements and software architecture:

- **Performance:** The architecture should be designed to localise critical operations within a small number of components if performance is a critical requirement. These components should all be deployed on the same computer rather than distributed across the network.

- **Security:** A layered structure for the architecture should be used if security is a critical requirement. The most critical assets should be protected in the innermost layers, with a high level of security validation applied to these layers.

- **Safety:** The architecture should be designed so that safety-related operations are all located in either a single component or in a small number of components if safety is a critical requirement. This reduces the costs and problems of safety validation and makes it possible to provide related protection systems that can safely shut down the system in the event of failure.

- **Availability:** The architecture should be designed to include redundant components so that it is possible to replace and update components without stopping the system.

- **Maintainability:** the system architecture should be designed using fine-grain, self-contained components that may be changed easily. The producers of data should be separated from consumers of data and shared data structures should be avoided.

Architectural Patterns

An architectural pattern is a description of a system organisation (Garlan and Shaw, 1993). These patterns capture the essence of an architecture that has been used in other software systems. An architectural pattern can be thought of as a stylised, abstract description of good practice, which has been tried and tested in different systems and environments.

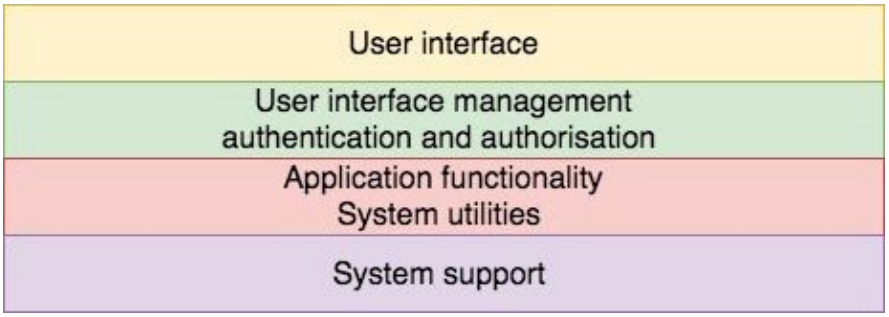
It is important that you are aware of common patterns, where they can be used, and their strengths and weaknesses when making architectural decisions. We will now discuss some frequently used patterns.

Layered Architecture

Fundamental to architectural design is the notions of separation and independence because they allow changes to be localised. The layered architecture pattern is a way of achieving this. With this pattern, the system

functionality is organised into separate layers, and each layer only relies on the facilities and services offered by the layer immediately below it.

Layered architecture allows a system to be developed incrementally. As each layer is developed, some of the services provided by that layer may be made available to the users. The architecture is also changeable and portable which means that as long as its interface does not change, a layer can be replaced by another, equivalent layer. Also, only the adjacent layer is affected when layer interfaces change or new facilities are added to a layer. Layered systems localise machine dependencies in inner layers which makes it easier to provide multiplatform implementations of an application system. Only the inner, layers, which are machine dependent, need be re-implemented to take account of the facilities of a different operating system or database. The diagram below shows an example of a layered architecture with four layers.



Example of a Layered Architecture

The lowest layer in a layered architecture includes system support software. This is typically database and operating system support. The next layer is the application layer which includes the components concerned with the application functionality as well as utility components that are used by other application components. The third layer is concerned with user interface management and providing user authentication and authorisation, while the top layer provides user interface facilities. Remember however that the number of layers does not matter and any of the layers could be split into two or more layers.

Below is a description of the layered architecture:

Description	Organises the system into layers with related functionality associated with each layer. A layer provides services to the layer above it. The lowest layers, therefore, represent core services that are likely to be used throughout the system.

When used	Used when building new facilities on top of existing systems. When the development is spread across a number of teams with each team responsible for a layer of functionality. When there is a requirement for multi-level security.
Advantages	As long as the interface is maintained, it allows replacement of entire layers. Redundant facilities (for example, authentication) can be provided in each layer to increase the dependability of the system.
Disadvantages	Providing a clean separation between layers is difficult to in practice and a high-level layer may have to interact directly with lower-level layers rather than through the layer immediately below it. Because of multiple levels of interpretation of a service request as it is processed at each layer, performance can be an issue.

Repository Architecture

The repository pattern describes how a set of interacting components can share data. Systems that use large amounts of data are often organised around a shared database or repository. This model is therefore suited to applications in which data is generated by one component and used by another.

Below is a description of the repository architecture:

Description	All data in a system is managed in a central repository that is accessible to all system components. Components do not interact directly, only through the repository.
When used	This pattern is used when you have a system in which large volumes of information are generated that has to be stored for a long time. It is also used in data-driven systems where the inclusion of data in the repository triggers an action or tool.
Advantages	Components can be independent, which means they do not need to know of the existence of other components. Changes made by one component can be propagated to all components. All data can be managed consistently as it is all in one place.

Disadvantages	The repository is a single point of failure so problems in the repository affect the whole system. They may be inefficiencies in organising all communication through the repository. Distributing the repository across several computers may be difficult.

An efficient way to share large amounts of data is to organise tools around a repository since there is no need to transmit data explicitly from one component to another. Components, however, must operate around an agreed repository data model. This is a compromise between the specific needs of each tool. It may, therefore, be difficult or impossible to integrate new components if their data models do not fit the agreed schema. Also, in practice, it may be difficult to distribute the repository over a number of machines. Although it is possible to distribute a logically centralised repository, there may be problems with data redundancy and inconsistency.

Client-Server Architecture

A system that follows the client-server pattern is organised as a set of services and associated servers, and clients that access and use the services. There are three major components in this model:

A set of servers that offer services to other components. For example, print servers that offer printing services, file servers that offer file management services, and a compile server, which offers programming language compilation services.

A set of clients that call on the services offered by servers. There will normally be several instances of a client program executing concurrently on different computers.

A network that allows the clients to access these services. Most client-server systems are implemented as distributed systems and connected using Internet protocols.

Below is a description of the client-server architecture:

Description	In this architecture, the functionality of the system is organised into services, with each service delivered from a separate server. Clients are users of these services and access servers to make use of them.

When used	Used when data in a shared database has to be accessed from a range of locations. May also be used when the load on a system is variable because servers can be replicated.
Advantages	Servers can be distributed across a network. General functionality (e.g., a printing service) can be available to all clients and does not need to be implemented by all services.
Disadvantages	Each service is a single point of failure and therefore is susceptible to denial of service attacks or server failure. It depends on the network as well as the system so performance may be unpredictable. If servers are owned by different organisations there may be management problems.

Pipe and Filter Architecture

The pipe and filter architecture is a model of the run-time organisation of a system where functional transformations process their inputs and produce outputs. Data flows from one to another and is transformed as it moves through the sequence and each processing step is implemented as a transform. Input data flows through these transforms until converted to output. The transformations may execute sequentially or in parallel. The data can be processed by each transform item by item or in a single batch.

Below is a description of the pipe and filter architecture:

Description	The processing of the data in a system is organised so that each processing component, or filter, is discrete and carries out one type of data transformation. The data flows, as in a pipe, from one component to another for processing.
When used	Used in data processing applications where inputs are processed in separate stages to generate related outputs.
Advantages	Easy to understand and supports transformation reuse. Workflow style matches the structure of many business processes. Evolution by adding transformations is straightforward. Can be implemented as either a sequential or concurrent system.
Disadvantages	The format for data transfer has to be agreed upon between communicating transformations. Each transformation must

	parse its input and un-parse its output to the agreed form. This increases system overhead and may mean that it is impossible to reuse functional transformations that use incompatible data structures.

Compulsory Task

Answer the following questions:

- When describing a system, explain why you may have to design the system architecture before the requirements specification is complete.
- Explain why design conflicts might arise when designing an architecture where availability and security requirements are the most important non-functional requirements.
- Suggest an architecture for a system (such as iTunes) that is used to sell and distribute music on the Internet. What architectural patterns are the basis for this architecture?
- Explain why you normally use several architectural patterns when designing the architecture of a large system. What additional information might be useful when designing large systems (apart from the information about patterns discussed in this task)?

Rate us

Share your thoughts

Hyperion strives to provide internationally-excellent course content that helps you achieve your learning outcomes.

Think the content of this task, or this course as a whole, can be improved or think we've done a good job?

Click here to share your thoughts anonymously.