Microkernel Pattern

Microkernel Introduction

- The Microkernel architectural pattern applies to software systems that must be able to adapt to changing system requirements.
- It separates a minimal functional core from extended functionality and customer specific parts.
- The microkernel also serves as a socket for plugging in these extensions and coordinating their collaboration.
- Microkernel systems have mainly been described in relation to the design of operating systems. However, this pattern is also applicable to other domains, e.g. financial applications and database systems.

Problem

- How to design an architecture for the development of several applications with similar programming interfaces that build on the same core functionality.

Context

- Developing software for an application domain that needs to cope with a broad spectrum of similar standards and technologies is a non-trivial task.
- Well-known examples are application platforms such as operating systems and graphical user interfaces.
- Such systems often have a long life-span, often ten years or more.
- Over time periods of this length, new technologies emerge and old ones change.

Context (2)

- Some applications exist in multiple versions.
  - Each version offers a different set of functionality to its users, or differs from other versions in specific aspects, such as its user interface.
  - Despite their differences, however, all versions of the application should be based on a common architecture and functional core.
- The goal is to avoid architectural drift between the versions of the application and to minimize development and maintenance effort for shared functionality.
- It should be easy to provide a particular application version with different user interfaces, and also to run the version on different platforms.

Forces

- The application platform must cope with continuous hardware and software evolution.
- The application platform should be portable, extensible and adaptable to allow easy integration of emerging technologies.
- The success of such application platforms may also depend on their capability to run applications written for existing standards.
- To support a broad range of applications, there is a need for more than one view of the functionality of the underlying application platform.
- Ideally, an application platform such as an operating system or a database should also be able to emulate other application platforms that belong to the same application domain.

Solution

- Compose different versions of the application by extending a common but minimal core via a “plug-and-play” infrastructure.
- The Microkernel pattern defines five kinds of participating components:
  - Microkernel
  - Internal servers
  - External servers
  - Adapters
  - Clients

Solution (2)

- The microkernel implements the fundamental functionality shared by all application versions and provides the infrastructure for integrating version specific functionality.
- Internal servers implement version specific core functionality, and external servers version specific user interfaces or APIs.
- A specific application version is configured by connecting the corresponding internal servers with the microkernel, and providing appropriate external servers to access its functionality.
- Consequently, all versions of the application share a common functional and infrastructural core, but provide a tailored function set and look-and-feel.

Solution (3)

- Clients, whether human or other software systems, access the microkernel's functionality solely via the interfaces or APIs provided by the external servers, which forward all requests they receive to the microkernel.
- If the microkernel implements the requested function itself, it executes the function, otherwise it routes the request to the corresponding internal server.
- Results are returned accordingly so that the external servers can display or deliver them to the client.

Solution (4)

- A problem arises if a client needs to access the interfaces of its external server directly.
  - Every communication with an external server must be hard-coded into the client code.
  - Such a tight coupling between clients and servers does not support changeability very well.
- We therefore introduce interfaces between clients and their external servers to protect clients from direct dependencies.
- Adapters represent these interfaces between clients and their external servers, and allow clients to access the services of their external server in a portable way.

Microkernel Details

- The internal structure of the microkernel is typically based on Layers.
  - The bottommost layer abstracts from the underlying system platform, thereby supporting the portability of all higher levels.
  - The second layer implements infrastructure functionality, such as resource management, on which the microkernel depends.
  - The layer above hosts the domain functionality that is shared by all application versions.
  - The topmost layer includes the mechanisms for configuring internal servers with the microkernel, as well as for routing requests from external servers to their intended recipient.
- The microkernel should be kept as small a s possible to reduce memory requirements and provide mechanisms that execute quickly.
  - Additional and more complex services are therefore implemented by internal servers that the microkernel activates or loads only when necessary.

Internal and External Server Details

- Internal servers can be considered extensions of the microkernel and are only accessible by the microkernel component.
- Internal servers may be implemented as separate processes or as shared libraries:
  - Graphics card drivers are developed as shared libraries because they only act on behalf of clients.
  - In contrast, page fault handlers are separate processes. They always have to remain in main memory and cannot be swapped to external storage.
- Internal servers follow a similar Layers design as the microkernel, but do not usually provide a routing layer.
- Each external server is implemented as a separate process that provides its own service interface. The internal architecture of an external server depends on the policies it comprises.

Microkernel Benefits

- *Portability.* A Microkernel system offers a high degree of portability, for two reasons:
  - In most cases you do not need to port external servers or client applications if you port the Microkernel system to a new software or hardware environment.
  - Migrating the microkernel to a new hardware environment only requires modifications to the hardware-dependent parts of the microkernel and does not affect external servers.
- *Flexibility and Extensibility.*
  - If you need to implement an additional view, all you need to do is add a new external server.
  - Extending the system with additional capabilities only requires the addition or extension of internal servers.

Microkernel Benefits (2)

- *Separation of policy and mechanism.*
  - The microkernel component provides all the *mechanisms* necessary to enable external servers to implement their *policies*.
  - This strict separation of policies and mechanisms increases the maintainability and changeability of the whole system.
  - It also allows you to add new external servers that implement their own specialized views. If the microkernel component were to implement policies, this would unnecessarily limit the views that could be implemented by external servers.

Microkernel Benefits (3)

- *Scalability*.
  - A distributed Microkernel system is applicable to the development of operating systems or database systems for computer networks, or multiprocessors with local memory.
  - If your Microkernel system works on a network of machines, it is easy to scale the Microkernel system to the new configuration when you add a new machine to the network.
- *Transparency*.
  - In a distributed system components can be distributed over a network of machines.
  - In such a configuration, the Microkernel architecture allows each component to access other components without needing to know their location. All details of inter-process communication with servers are hidden from clients by the adapters and the microkernel.

Microkernel Liabilities

- *Performance*.
  - A monolithic software system designed to offer a specific view will generally have better performance than a Microkernel system supporting different views.
  - We therefore have to pay a price for flexibility and extensibility.
  - If the communication within the Microkernel system is optimized for performance, however, this price can be small.
- *Complexity of design and implementation*.
  - Developing a Microkernel-based system is a non-trivial task.
  - For example, it can sometimes be very difficult to analyze or predict the basic mechanisms a microkernel component must provide.
  - In addition, the separation between mechanisms and policies requires in-depth domain knowledge and considerable effort during requirements analysis and design.