

Structure The *Microkernel* pattern defines five kinds of participating components, namely *internal servers*, *external servers*, *adapters*, *clients* and the *microkernel*.

The *microkernel* represents the main component within the pattern. It implements central services such as communication facilities or resource handling. Other components build upon all or some of these basic services. They do this indirectly by using one or more interfaces which comprise the functionality the microkernel exposes.

Many system-specific dependencies are encapsulated within the microkernel. For instance, most of the hardware dependent parts are hidden from other participants. Clients of the microkernel only see common abstractions of the underlying application domain and of the platform specifics.

Furthermore, the microkernel is responsible for maintaining system resources such as processes or files. It controls and coordinates the access to these resources.

To summarize, a microkernel implements atomic services which we call mechanisms. These mechanisms serve as a fundamental base upon which more complex functionality, so-called policies, are composed.

Class Microkernel	Collaborators • Internal Server
Responsibility <ul style="list-style-type: none"> • Provides core mechanisms. • Offers communication mechanisms. • Encapsulates system dependencies. • Manages and 	

➔ Let us recapture the motivating example: Among other operating systems we want to support UNIX System V and OS/2 Warp in Hydra. When implementing Hydra's process

model, we face a problem. A system call such as creating a new child process is implemented in UNIX by cloning an existing process with copying the whole address space. OS/2 Warp handles process creation totally different in that it does not copy the address space of the parent process. In other words, OS/2 Warp and UNIX offer different policies regarding to processes. Thus, Hydra is designed to supply basic services such as mechanisms to create processes as well as mechanisms for cloning existing process spaces. These are combined in various ways for implementing both, the process model of UNIX System V and the process model of OS/2 Warp. □

An *internal server*—also known as *subsystem*—extends the functionality the microkernel provides. It represents a separate component offering additional functionality. The microkernel invokes the functionality of internal servers due to service requests. Thus, internal servers may also encapsulate some system dependencies. For instance, drivers that support specific graphics cards are good candidates for internal servers.

One of the design goals is to keep the microkernel as small as possible in order to reduce memory contention. Another goal is to provide mechanisms in such a way that they can be executed very fast to reduce service execution time. Additional and more complex services are implemented by internal servers which are activated or loaded by the microkernel. Hence, we may consider internal servers as extensions of the microkernel. Note, that internal servers are only accessible by the microkernel component.

<p>Class Internal Server</p>	<p>Collaborators • Microkernel</p>
<p>Responsibility • Implements additional services. • Encapsulates some system specifics.</p>	

An *external server*—also known as *personality*—is a component that uses the microkernel for implementing its own view of the underlying application domain. In this context, a view denotes a layer of abstraction that is built on top of the atomic services the microkernel provides. Different external servers provide different policies with respect to the application domain under consideration.

Similar to the microkernel, external servers expose their functionality by exporting interfaces. They receive service requests from client applications, interpret these, execute the appropriate services and return results back to the callers. Since the implementation of services relies on microkernel mechanisms, external servers need to access the microkernel's programming interfaces.

<p>Class External Server</p>	<p>Collaborators • Microkernel</p>
<p>Responsibility • Provides programming interfaces for its clients.</p>	

➡ In Hydra you are going to implement an OS/2 Warp external server as well as a UNIX System V external server. Both of these servers use the mechanisms of the underlying microkernel to implement a complete set of OS/2 Warp and UNIX System V system calls. □

A *client* is an application that is associated with exactly one external server. It does only access the programming interfaces the external server provides.

If a client would need to access the interfaces of its external server directly, a problem arises: For interoperating with the external servers, each client would have to utilize the available communication facilities. Thus, every communication with an external server must be hard-coded into the client code. Such a tight coupling between clients

and servers, however, leads to various disadvantages: Firstly, such a system does not support changeability very well. And secondly, if external servers emulate existing application platforms, client applications developed for these platforms will not run without previous modifications.

Therefore, interfaces between clients and their external servers are introduced to protect clients from all those system dependencies:

Adapters—also known as *emulators*—represent interfaces between clients and their external servers. They allow clients to access the services of their external server. If the external server implements an existing application platform, the corresponding adapter mimics the programming interfaces of that platform. Thus, clients written for the emulated platform can be compiled and run without any modifications. In addition, adapters protect clients from the microkernel specifics.