BITS Pilani

Pilani|Dubai|Goa|Hyderabad

Module 7

Patterns – Part 3 Suppli

Adaptable Systems.

Harvinder S Jabbal
SSZG653 Software Architectures

Categories

From Mud to Structure.

- Patterns in this category help you to avoid a 'sea' of components or objects.
- In particular, they support a controlled decomposition of an overall system task into cooperating subtasks.
- The category includes
  - the Layers pattern
  - the Pipes and Filters pattern
  - the Blackboard pattern

Distributed Systems.

- This category includes one pattern.
  - Broker
- and refers to two patterns in other categories,
  - Microkernel
  - Pipes and Filters
- The Broker pattern provides a complete infrastructure for distributed applications.
- The Microkernel and Pipes and Filters patterns only consider distribution as a secondary concern and are therefore listed under their respective primary categories.

Interactive Systems.

- This category comprises two patterns,
  - the Model-View-Controller pattern (well-known from Smalltalk,)
  - the Presentation-Abstraction-Control pattern.
- Both patterns support the structuring of software systems that feature human-computer interaction.

Adaptable Systems.

- This category includes
  - The Reflection pattern
  - the Microkernel pattern
- strongly support extension of applications and their adaptation to evolving technology and changing functional requirements.

Adaptable Systems.

- This category includes
  - The Reflection pattern
  - the Microkernel pattern
- strongly support extension of applications and their adaptation to evolving technology and changing functional requirements.

Adaptable

- Systems evolve over time
  - -new functionality is added and existing services are changed.
- They must support new versions of
  - operating systems,
  - user-interface platforms or
  - third-party components and libraries.
- Adaptation to
  - new standards or
  - hardware platforms may also be necessary.
- During system design and implementation,
  - customers may request new features, often urgently and at a late stage. Y
  - You may also need to provide services that differ from customer to customer.

Design for Change

- Design for change is therefore a major concern when specifying the architecture of a software system.
- An application should support its own modification and extension a priori.
- Changes should not affect the core functionality or key design abstractions, otherwise the system will be hard to maintain and expensive to adapt to changing requirements.

innovate achieve lead

BITS Pilani

Pilani|Dubai|Goa|Hyderabad

Adaptable Systems: Microkernel

Adaptable Systems: Microkernel

- 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.

Context

The development of

several applications

that use similar programming interfaces that build on the

same core functionality.

Problem

- 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.

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 applications in your domain need to support different, but similar, application platforms.
- The applications may be categorized into groups that use the same functional core in different ways, requiring the underlying application platform to emulate existing standards.
- The functional core of the application platform should be separated into a component with minimal memory size, and services that consume as little processing power as possible.

Solution

- Encapsulate the fundamental services of your application platform in a microkernel component.
- The microkernel includes functionality that enables other components running in separate processes to communicate with each other.
- It is also responsible for maintaining system- wide resources such as files or processes.
- In addition, it provides interfaces that enable other components to access its functionality.

Solution - details

- Core functionality that cannot be implemented within the microkernel without unnecessarily increasing its size or complexity should be separated in internal servers.
- External servers implement their own view of the underlying microkernel.
- To construct this view, they use the mechanisms available through the interfaces of the microkernel. Every external server is a separate process that itself represents an application platform.
- Hence, a Microkernel system may be viewed as an application platform that integrates other application platforms.
- Clients communicate with external servers by using the communication facilities provided by the microkernel.

Structure

The Microkernel pattern defines five kinds of participating components:

- Internal servers
- External servers
- Adapters
- Clients
- Microkernel

Microkernel

- represents the main component of the pattern.
- It implements central services such as communication facilities or resource handling.
- Other components build on all or some of these basic services.
- They do this indirectly by using one or more interfaces that comprise the functionality exposed by the microkernel.

Microkernel Component

- A microkernel implements **atomic** services, which we refer to as **mechanisms**.
- These **mechanisms** serve as a fundamental base on which more complex functionality, called **policies**, are constructed.
- The microkernel is also responsible for maintaining **system resources** such as **processes** or **files**.
- It controls and coordinates the access to these resources.

Class Microkernel	***Collaborators***
Responsibility • Provides core mechanisms. • Offers communication facilities. • Encapsulates system dependencies. • Manages and controls resources.	• Internal Server

Internal Server Component

- Also known as a subsystem.
- They are extensions of the microkernel.
- Only accessible by the microkernel component.
- Extends the functionality provided by the microkernel.
- It is a separate component that offers additional functionality.
- The microkernel invokes the functionality of internal servers via service requests.
- Internal servers can therefore encapsulate some dependencies on the underlying hardware or software system.
- For example, device drivers that support specific graphics cards are good candidates for internal servers.

Class Internal Server	***Collaborators***
Responsibility • Implements additional services. • Encapsulates some system specifics.	• Microkernel

External Server Component

- also known as a personality
- uses the microkernel for implementing its own view of the underlying application domain.
- a view denotes a layer of abstraction built on top of the atomic services provided by the microkernel.
- Different external servers implement different policies for specific application domains.
- External servers expose their functionality by exporting interfaces in the same way as the microkernel itself does.
- Each of these external servers runs in a separate process.
- It receives service requests from client applications, uses the communication facilities provided by the microkernel, interprets these requests, executes the appropriate services and returns results to its clients.
- The implementation of services relies on microkernel mechanisms, so external servers need to access the microkernel's programming interfaces.

Class External Server	***Collaborators***
Responsibility • Provides programming interfaces for its clients.	• Microkernel

Client & Adapter

If the external server implements an existing application platform, the corresponding adapter mimics the programming interfaces of that platform.

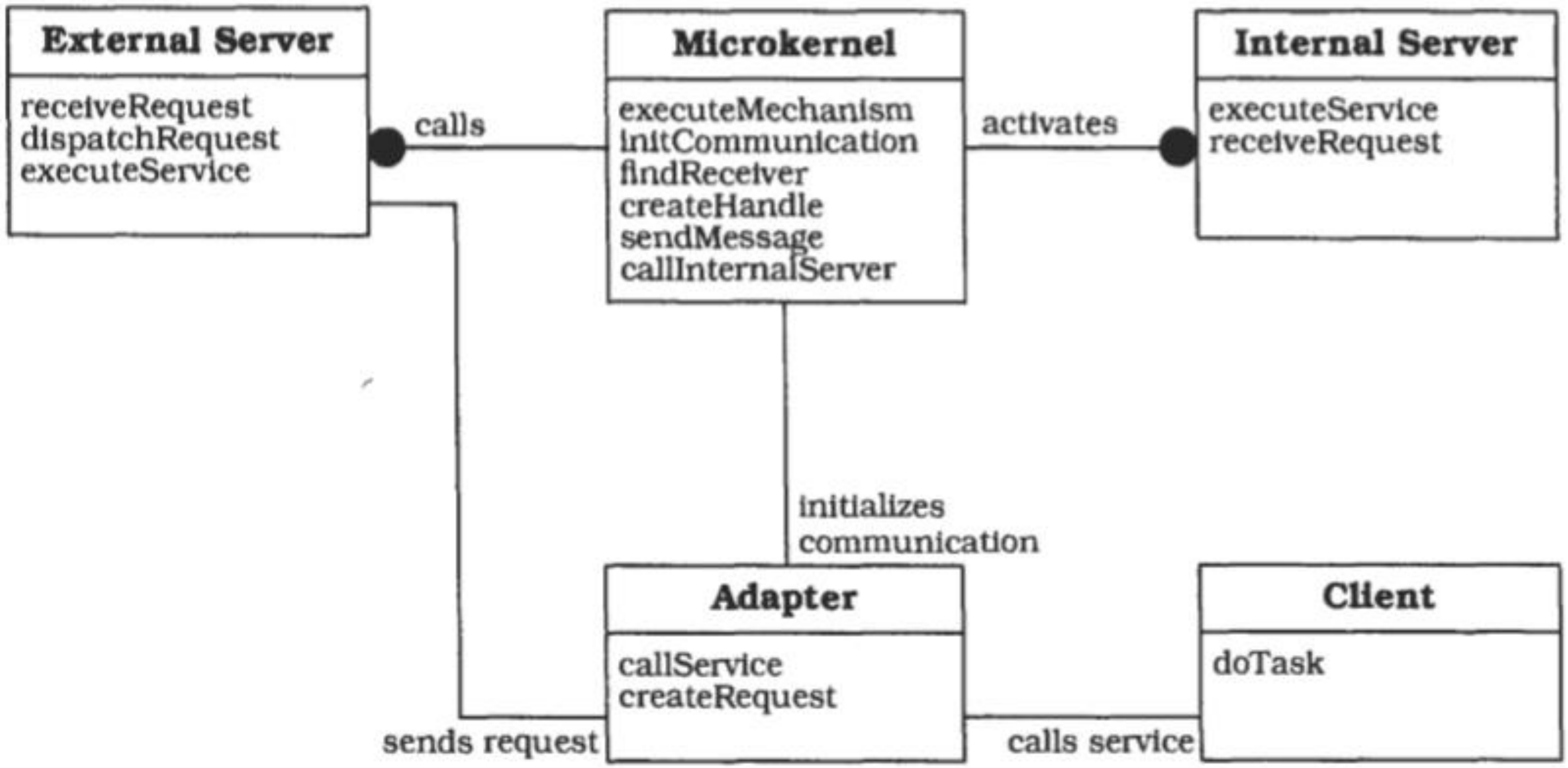
Adapters

- also known as emulators
- represent these interfaces between clients and their external servers
- allow clients to access the services of their external server in a portable way

Class Client	*Collaborators*
Responsibility • Represents an application.	• Adapter

Class Adapter	*Collaborators*
Responsibility • Hides system dependencies such as communication facilities from the client. • Invokes methods of external servers on behalf of clients.	• External Server • Microkernel

OMT Diagram


External Server
receiveRequest
dispatchRequest
executeService
calls
Microkernel
executeMechanism
initCommunication
findReceiver
createHandle
sendMessage
callInternalServer
activates
Internal Server
executeService
receiveRequest
initializes
communication
Adapter
callService
createRequest
sends request
Client
doTask
calls service

Implementation

1. Analyze the application domain.
2. Analyze external servers
3. Categorize the services.
4. Partition the categories.
5. Find a consistent and complete set of operatims and abstractims for every category.
6. Determine strategies for request transmission and retrieval.
7. Structure the microkernel component.
8. specify the programming interface
9. The microkernel is responsible for managing all system resources such as memory blocks, devices or device contexts-a handle to an output area in a graphical user interface implementation.
10. Design and implement the internal ser-uers as separate processes or shared libraries.
11. Implement the external servers
12. Implement the adapters
13. Develop client applications or use existing ones for the ready-to-run Microkernel system

Variant:

Microkernel System with indirect Client-Server connections.

- A client that wants to send a request or message to an external server asks the microkernel for a communication channel.
- After the requested communication path has been established, client and server communicate with each other indirectly using the microkernel as a message backbone.
- Using this variant leads to an architecture in which all requests pass through the microkernel.
- You can apply it, for example, when security requirements force the system to control all communication between participants.

Variant: Distributed Microkernel System

- A microkernel can also act as a message backbone responsible for sending messages to remote machines or receiving messages from them.
- Every machine in a distributed system uses its own microkernel implementation.
- From the user's viewpoint the whole system appears as a single Microkernel system-the distribution remains transparent to the user.
- A distributed Microkernel system allows you to distribute servers and clients across a network of machines or microprocessors.
- To achieve this the micro kernels in a distributed implementation must include additional services for communicating with each other.

Benefits

1. Portability
2. Flexibility and Extensibility
3. Separation of policy and mechanism
4. Scalability
5. Reliability
6. Transparency

liability

1. Performance.
2. Complexity of design and implementation.

innovate achieve lead

BITS Pilani

Pilani|Dubai|Goa|Hyderabad

Adaptable Systems: Reflection

Adaptable Systems: Reflection

- The Reflection architectural pattern provides a mechanism for changing structure and behaviour of software systems dynamically.
- It supports the modification of fundamental aspects, such as type structures and function call mechanisms.
- In this pattern, an application is split into two parts.
    - A meta level provides information about selected system properties and makes the software self-aware.
    - A base level includes the application logic.
- Its implementation builds on the meta level.
- Changes to information kept in the meta level affect subsequent base-level behaviour.

Context

- Building systems that support their own modification a priori.

Problem

- Software systems evolve over time.
- They must be open to modifications in response to changing technology and requirements.
- Designing a system that meets a wide range of different requirements a priori can be an overwhelming task.
- A better solution is to specify an architecture that is open to modification and extension.
- The resulting system can then be adapted to changing requirements on demand.
- In other words, we want to design for change and evolution.

Forces

- Changing software is tedious, error prone, and often expensive.
- Adaptable software systems usually have a complex inner structure.
- The more techniques that are necessary for keeping a system changeable, such as parameterization, subclassing, mix-ins, or even copy and paste, the more awkward and complex its modification becomes.
- Changes can be of any scale, from providing shortcuts for commonly-used commands to adapting an application framework for a specific customer.
- Even fundamental aspects of software systems can change, for example the communication mechanisms between components.

Solution

- Make the software self-aware, and make selected aspects of its structure and behaviour accessible for adaptation and change.
- This leads to an architecture that is split into two major parts:
  - a meta level and
  - a base level.

Meta Level

- The meta level provides a self-representation of the software to give it knowledge of its own structure and behavior, and consists of so-called rnetaobjects.
- Metaobjects encapsulate and represent information about the software.
- Examples include type structures, algorithms, or even function call mechanisms.

Base Level

- The base level defines the application logic.
- Its implementation uses the metaobjects to remain independent of those aspects that are likely to change.
- For example, base-level components may only communicate with each other via a metaobject that implements a specific user-defined function call mechanism.
- Changing this metaobject changes the way in which base-level components communicate, but without modifing the base-level code.

metaobject protocol (MOP)

- An interface is specified for manipulating the metaobjects.
- It is called the metaobject protocol (MOP), and allows clients to specify particular changes, such as modification of the function call mechanism metaobject mentioned above.
- The metaobject protocol itself is responsible for checking the correctness of the change specification, and for performing the change.
- Every manipulation of metaobjects through the metaobject protocol affects subsequent base-level behavior, as in the function call mechanism example.

Structure

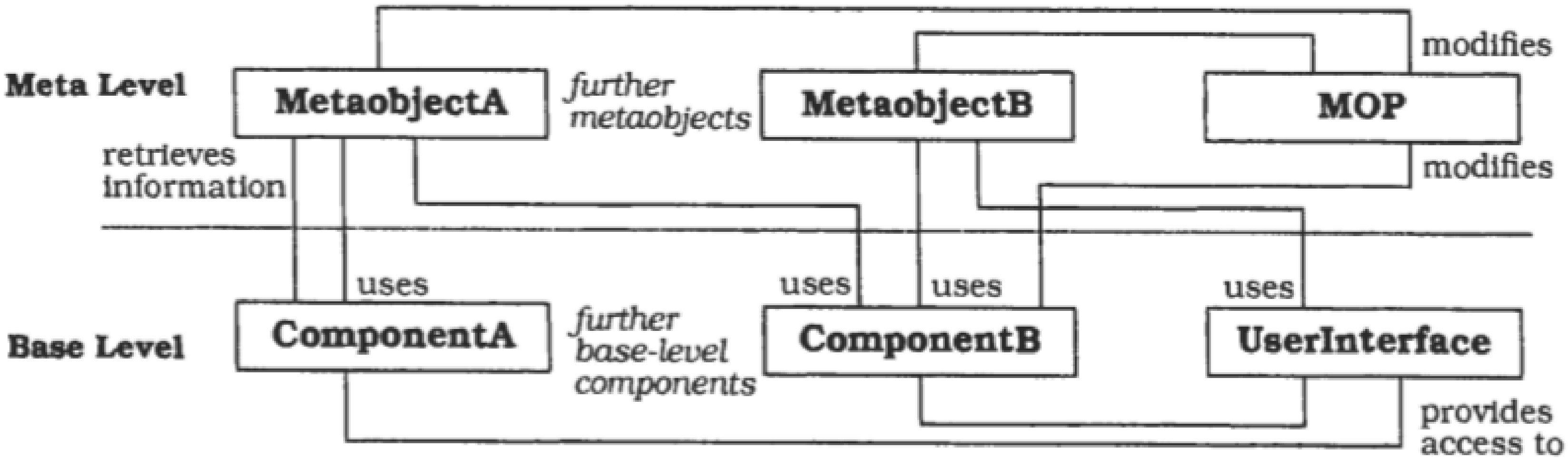
Class: Base Level	Collaborators
Responsibility • Implements the application logic. • Uses information provided by the meta level.	• Meta Level

Class: Metaobject Protocol	Collaborators
Responsibility • Offers an interface for specifying changes to the meta level. • Performs specified changes	• Meta Level • Base Level

Class: Meta Level	Collaborators
Responsibility • Encapsulates system internals that may change. • Provides an interface to facilitate modifications to the meta-level.	• Base Level

- The meta level consists of a set of metaobjects. Each metaobject encapsulates selected information about a single aspect of the structure, behaviour, or state of the base level.
- The base level models and implements the application logic of the software.
- The metaobject protocol (MOP] serves as an external interface to the meta level, and makes the implementation of a reflective system accessible in a defined way.

OMT Diagram



- Since the base-level implementation explicitly builds upon information and services provided by metaobjects, changing them has an immediate effect on the subsequent behaviour of the base level.
- The general structure of a reflective architecture is very much like a Layered system

Implementation

1. Define a model of the application.
2. Identify varying behavior
3. Identify structural aspects of the system, which, when changed, should not affect the implementation of the base level
4. Identify system services that support both the variation of application services and the independence of structural details
5. Define the metaobjects.
6. Define the metaobject protocol.
7. Dew the base level.

Benefits

1. No explicit mod$cation of source code.
2. Changing a software system is easy
3. Support for many kinds of change

liabilities

1. Modifications at the meta level may cause damage
2. Increased number of components
3. Lower efficiency.
4. Not all potential changes to the software are supported.
5. Not all languages support refection.

Thank you

Credits: Material sourced from Text Book…

PATTERN-ORIENTED SOFTWARE ARCHITECTURE

A System of Patterns

- Frank Buschmann
- Regine Meunier
- Hans Rohnert
- Peter Sornmerlad
- Michael Stal

of Siemens AG, Germany