FUTURE VISION BIE

One Stop for All Study Materials

& Lab Programs

Future Vision

By K B Hemanth Raj

Scan the QR Code to Visit the Web Page

Or

Visit : https://hemanthrajhemu.github.io

Gain Access to All Study Materials according to VTU,
CSE – Computer Science Engineering,
ISE – Information Science Engineering,
ECE - Electronics and Communication Engineering
& MORE...

Join Telegram to get Instant Updates: https://bit.ly/VTU_TELEGRAM

Contact: MAIL: futurevisionbie@gmail.com

INSTAGRAM: www.instagram.com/hemanthraj_hemu/

INSTAGRAM: www.instagram.com/futurevisionbie/

WHATSAPP SHARE: https://bit.ly/FVBIESHARE

UNIT 6
ARCHITECTURAL PATTERNS-3

ADAPTABLE SYSTEMS

The systems that evolve over time - new functionality is added and existing services are changed are called adaptive systems.
They must support new versions of OS, user-interface platforms or third-party components and libraries.
Design for change is a major concern when specifying the architecture of a software system.
We discuss two patterns that helps when designing for change.

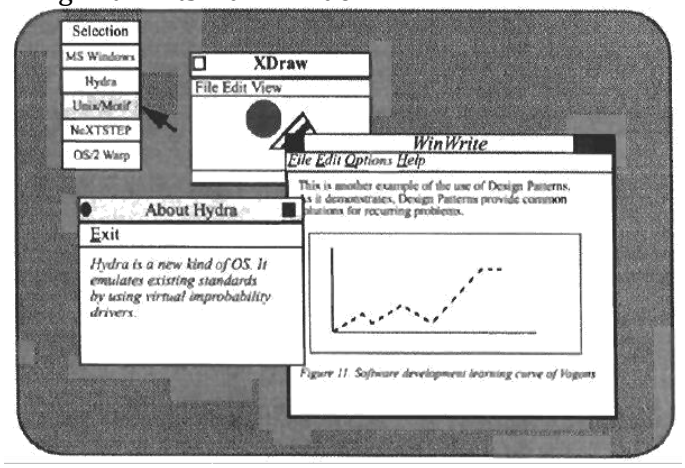
- ***Microkernel pattern*** → applies to software systems that must be able to adapt to changing system requirements.
- ***Reflection pattern*** → provides a mechanism for changing structure and behavior of software systems dynamically.

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.

Example:

Suppose we intend to develop a new operating system for desktop computers called Hydra. One requirement is that this innovative operating system must be easily portable to the relevant hardware platforms, and must be able to accommodate future developments easily. It must also be able to run applications written for other popular operating systems such as Microsoft Windows and UNIX System V. A user should be able to choose which operating system he wants from a pop-up menu before starting an application. Hydra will display `all` the applications currently running within its main window:



Context:

The development of several applications that use similar programming interfaces that build on the same care functionality.

Problem:

Developing software for an application domain that needs to cope with a broad spectrum of similar standards and technology is a nontrivial task well known. Ex: are application platform such as OS and GUI'S. The following forces are to be considered when designing such systems.

- 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.

Application platform such as an OS should also be able to emulate other application platforms that belong to the same application domain. This leads to following forces.

- ♠ 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.

Additional force must be taken to avoid performance problem and to guarantee scalability.

- 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:

- Encapsulates the fundamental services of your applications platform in a microkernel component.
- It includes functionality that enables other components running in different process to communicate with each other.
- It provides interfaces that enable other components to access its functionality.
  - Core functionality should be included in ***internal servers.***
  - ***External servers*** implement their own view of the underlying microkernel.
  - ***Clients*** communicate with external servers by using the communication facilities provided by microkernel.

Structure:

Microkernel pattern defines 5 kinds of participating components.

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

❖ **Microkernel**

- The microkernel represents the main component of the pattern.
- It implements central services such as communication facilities or resource handling.
- The microkernel is also responsible for maintaining system resources such as processes or files.
- It controls and coordinates the access to these resources.
- 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.

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

❖ **An internal server (subsystem)**

- Extends the functionality provided by microkernel.
- It represents a separate component that offers additional functionality.
- Microkernel invokes the functionality of internal services via service requests.
- Therefore internal servers can encapsulates some dependencies on the underlying hardware or software system.

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

❖ **An external server (personality)**
- Uses the microkernel for implementing its own view of the underlying application domain.
- Each external server runs in separate process.
- It receives service requests from client applications using the communication facilities provided by the microkernel, interprets these requests, executes the appropriate services, and returns its results to clients.
- Different external servers implement different policies for specific application domains.

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

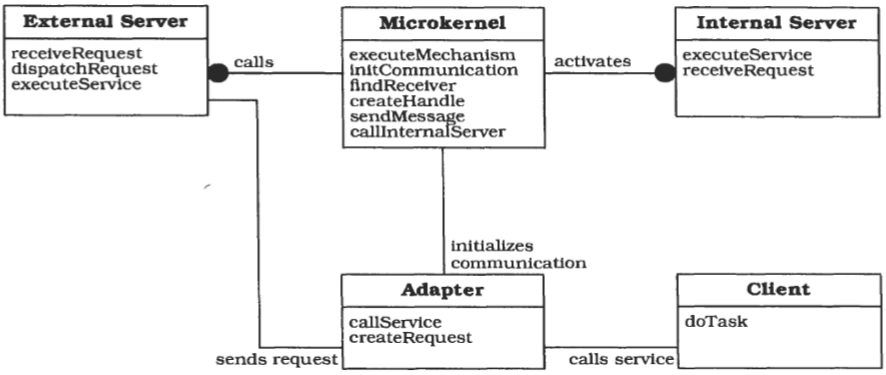
❖ **Client:**
- It is an application that is associated with exactly one external server. It only accesses the programming interfaces provided by the external server.
- Problem arises if a client accesses the interfaces of its external server directly (direct dependency)
  - ✓ Such a system does not support changeability
  - ✓ If ext servers emulate existing application platforms clients will not run without modifications.

❖ **Adapter** (emulator)
- Represents the interfaces b/w clients and their external servers and allow clients to access the services of their external server in a portable way.
- They are part of the clients address space.
- The following OMT diagram shows the static structure of a microkernel system.

Class Client	Collaborators • Adapter
Responsibility • Represents an application.	

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

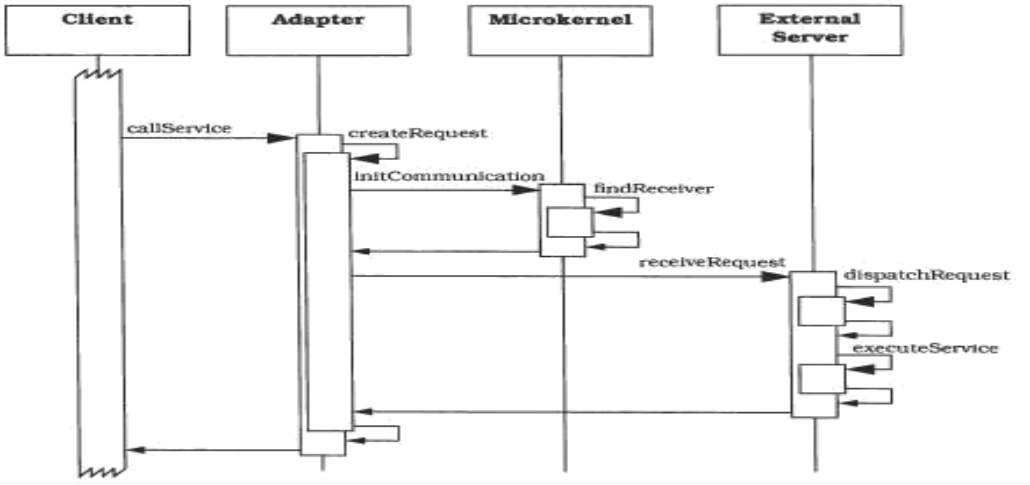
The following OMT diagram shows the static structure of a Microkernel system.



Dynamics:

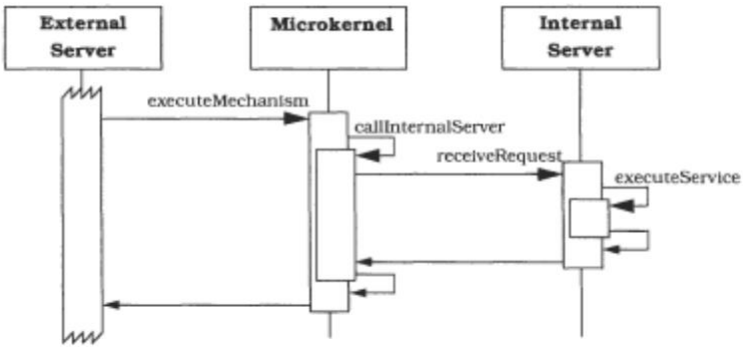
Scenario I demonstrates the behavior when a client calls a service of its external server:

- ★ At a certain point in its control flow the client requests a service from an external server by calling the adapter.
- ★ The adapter constructs a request and asks the microkernel for a communication link with the external server.
- ★ The microkernel determines the physical address of the external server and returns it to the adapter.
- ★ After retrieving this information, the adapter establishes a direct communication link to the external server.
- ★ The adapter sends the request to the external server using a remote procedure call.
- ★ The external server receives the request, unpacks the message and delegates the task to one of its own methods. After completing the requested service, the external server sends all results and status information back to the adapter.
- ★ The adapter returns to the client, which in turn continues with its control flow.



Scenario II illustrates the behavior of a Microkernel architecture when an external server requests a service that is provided by an internal server

- ★ The external server sends a service request to the microkernel.
- ★ A procedure of the programming interface of the microkernel is called to handle the service request. During method execution the microkernel sends a request to an internal server.
- ★ After receiving the request, the internal server executes the requested service and sends all results back to the microkernel.
- ★ The microkernel returns the results back to the external server.
- ★ Finally, the external server retrieves the results and continues with its control flow.



Implementation:

1. **Analyze the application domain:**
   Perform a domain analysis and identify the core functionality necessary for implementing ext servers.
2. **Analyze external servers**
   That is polices ext servers are going to provide
3. **Categorize the services**
   Group all functionality into semantically independent categories.
4. **Partition the categories**
   Separate the categories into services that should be part of the microkernel, and those that should be available as internal servers.
5. **Find a consistent and complete set of operations and abstractions** for every category you identified in step 1.
6. **Determine the strategies for request transmission and retrieval.**
   - Specify the facilities the microkernel should provide for communication b/w components.
   - Communication strategies you integrate depend on the requirements of the application domain.
7. **Structure the microkernel component**
   Design the microkernel using the layers pattern to separate system-specific parts from system-independent parts of the kernel.
8. **Specify the programming interfaces of microkernel**
   To do so, you need to decide how these interfaces should be accessible externally.
   You must take into an account whether the microkernel is implemented as a separate process or as a module that is physically shared by other component in the latter case, you can use conventional method calls to invoke the methods of the microkernel.
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 GUI implementation.
10. **Design and implement the internal servers as separate processes or shared libraries**
    - Perform this in parallel with steps 7-9, because some of the microkernel services need to access internal servers.
    - It is helpful to distinguish b/w active and passive servers ✓
      Active servers are implemented as processes
      ✓ Passive servers as shared libraries.
    - Passive servers are always invoked by directly calling their interface methods, where as active server process waits in an event loop for incoming requests.
11. **Implement the external servers**
    - Each external server is implemented as a separate process that provide its own service interface
    - The internal architecture of an external server depends on the policies it comprises
    - Specify how external servers dispatch requests to their internal procedures.
12. **Implement the adapters**
    - Primary task of an adapter is to provide operations to its clients that are forwarded to an external server.
    - You can design the adapter either as a conventional library that is statically linked to client during complication or as a shared library dynamically linked to the client on demand.

13. **Develop client applications**
    or use existing ones for the ready-to-run microkernel system.

Example resolved:

Shortly after the development of Hydra has been completed, we are asked to integrate an external server that emulates the Apple MacOS operating system. To provide a MacOS emulation on top of Hydra, the following activities are necessary:

- ✓ *Building an external server* on top of the Hydra microkernel that implements all the programming interfaces provided by MacOS, including the policies of the Macintosh user interface.
- ✓ *Providing an adapter* that is designed as a library, dynamically linked to clients.
- ✓ *Implementing the internal servers* required for MacOS.

Variants:

- *Microkernel system with indirect client-server communications.*
  In this variant, a client that wants to send a request or message to an external server asks the microkernel for a communication channel.
- *Distributed microkernel systems.*
  In this variant a microkernel can also act as a message backbone responsible for sending messages to remote machines or receiving messages from them.

Known uses:

- The **Mach** microkernel is intended to form a base on which other operating systems can be emulated. One of the commercially available operating systems that use Mach as its system kernel is NeXTSTEP.
- The operating system **Amoeba** consists of two basic elements: the microkernel itself and a collection of servers (subsystems) that are used to implement the majority of Amoeba's functionality. The kernel provides four basic services: the management of processes and threads, the low-level-management of system memory, communication services, both for point-to-point communication as well as group-communication, and low-level I/O services.
- **Chorus** is a commercially-available Microkernel system that was originally developed specifically for real-time applications.
- **Windows NT** was developed by Microsoft as an operating system for high-performance servers. **MKDE**
- (Microkernel Datenbank Engine) system introduces an architecture for database engines that follows the Microkernel pattern.

Consequences:

The microkernel pattern offers some important *Benefits:*

- **Portability:**
  High degree of portability
- **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 additional or extension of internal servers.
- **Separation of policy and mechanism**
  The microkernel component provides all the mechanisms necessary to enable external servers to implement their policies.

Distributed microkernel has further benefits:

- ♠ **Scalability**
  A distributed Microkernel system is applicable to the development of operating systems or database systems for computer networks, or multiprocessors with local memory
- ♠ **Reliability**
  A distributed Microkernel architecture supports availability, because it allows you to run the same server on more than one machine, increasing availability. Fault tolerance may be easily supported because distributed systems allow you to hide failures from a user.

- **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.

The microkernel pattern also has *Liabilities:*

- **Performance:**
  Lesser than monolithic software system therefore we have to pay a price for flexibility and extensibility.
- **Complexity of design and implementation:**
  Develop a microkernel is a non-trivial task.

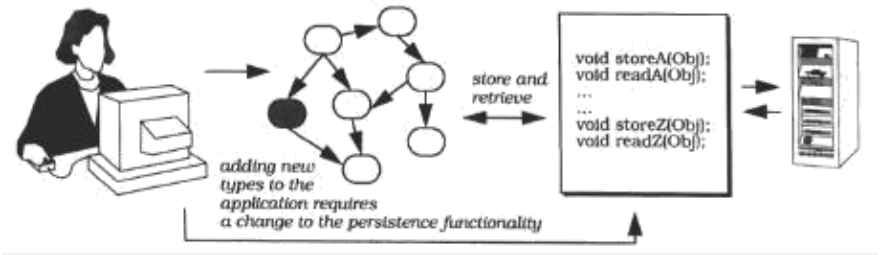
REFLECTION

The reflection architectural pattern provides a mechanism for changing structure and behavior of software systems dynamically. It supports the modification of fundamental aspects, such as the 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 s/w self aware.*
- *A base level includes application logic changes to information kept in the Meta level affect subsequent base-level behavior.*

Example:

Consider a C++ application that needs to write objects to disk and read them in again. Many solutions to this problem, such as implementing type-specific store and read methods, are expensive and error-prone. Persistence and application functionality are strongly interwoven. Instead we want to develop a persistence component that is independent of specific type structures



Context:

Building systems that support their own modification a prior

Problem:

- Designing a system that meets a wide range of different requirements a prior can be an overwhelming task.
- A better solution is to specify an architecture that is open to modification and extension i.e., we have to design for change and evolution.
- Several forces are associated with the problem:
  - ✓ Changing software is tedious, error prone and often expensive.
  - ✓ Adaptable software systems usually have a complex inner structure. Aspects that are subject to change are encapsulated within separate components.
  - ✓ The more techniques that are necessary for keep in a system changeable the more awkward and complex its modifications 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 ex. communication mechanisms b/w components.

Solution:

★ Make the software self-aware, and make select aspects of its structure and behavior accessible for adaptation and change.
- This leads to an architecture that is split into two major parts: A Meta level
- A base level

★ Meta level provides a self representation of the s/w to give it knowledge of its own structure and behavior and consists of so called *Meta objects* (they encapsulate and represent information about the software). Ex: type structures algorithms or function call mechanisms.

★ Base level defines the application logic. Its implementation uses the Meta objects to remain independent of those aspects that are likely to change.

★ An interface is specified for manipulating the Meta objects. It is called the *Meta object protocol* (MOP) and allows clients to specify particular changes.

Structure:

♥ Meta level
♥ Meta objects protocol(MOP)
♥ Base level

❖ **Meta level**

✓ Meta level consists of a set of Meta objects. Each Meta object encapsulates selected information about a single aspect of a structure, behavior, or state of the base level.
There are three sources for such information.
- It can be provided by run time environment of the system, such as C++ type identification objects.
- It can be user defined such as function call mechanism
- It can be retrieved from the base level at run time.

✓ All Meta objects together provide a self representation of an application.

✓ What you represent with Meta objects depends on what should be adaptable only system details that are likely to change r which vary from customer to customer should be encapsulated by Meta objects.

✓ The interface of a Meta objects allows the base level to access the information it maintains or the services it offers.

❖ **Base level**

✓ It models and implements the application logic of the software. Its component represents the various services the system offers as well as their underlying data model.

✓ It uses the info and services provided by the Meta objects such as location information about components and function call mechanisms. This allows the base level to remain flexible.

✓ Base level components are either directly connected to the Meta objects and which they depend or submit requests to them through special retrieval functions.

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

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

❖ **Meta object protocol (MOP)**

✓ Serves an external interface to the Meta level, and makes the implementation of a reflective system accessible in a defined way.

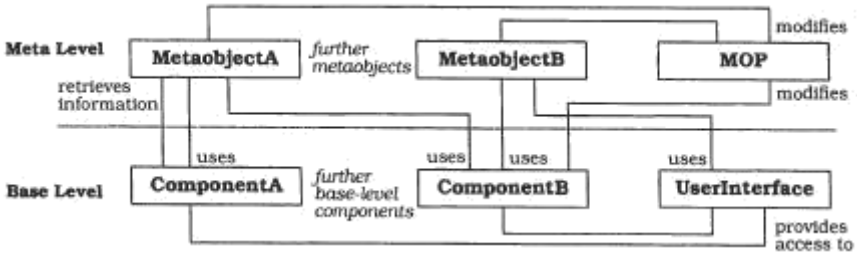
✓ Clients of the MOP can specify modifications to Meta objects or their relationships using the base level

- ✓ MOP itself is responsible for performing these changes. This provides a reflective application with explicit control over its own modification.
- ✓ Meta object protocol is usually designed as a separate component. This supports the implementation of functions that operate on several Meta objects.
- ✓ To perform changes, the MOP needs access to the internals of Meta objects, and also to base level components (sometimes).

  One way of providing this access is to allow the MOP to directly operate on their internal states. Another way (safer, inefficient) is to provide special interface for their manipulation, only accessible by MOP.

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

The general structure of a reflective architecture is very much like a Layered system



Dynamics:

Interested students can refer text book for scenarios since the diagrams are too hectic & can't be memorized

Implementation:

Iterate through any subsequence if necessary.

1. **Define a model of the application**
   Analyze the problem domain and decompose it into an appropriate s/w structure.
2. **Identify varying behavior**
   - ✓ Analyze the developed model and determine which of the application services may vary and which remain stable.
   - ✓ Following are ex: of system aspects that often vary ○ Real time constraints
     - ○ Transaction protocols
     - ○ Inter Process Communication mechanism
     - ○ Behavior in case of exceptions
     - ○ Algorithm for application services.
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 identified** In step 2 and the independence of structural details identified in step 3
   Eg: for system services are
   - ✓ Resource allocation
   - ✓ Garbage allocation
   - ✓ Page swapping
   - ✓ Object creation.
5. **Define the meta objects**
   - ✓ For every aspect identified in 3 previous steps, define appropriate Meta objects.

✓ Encapsulating behavior is supported by several domain independent design patterns, such as objectifier strategy, bridge, visitor and abstract factory.

6. Define the MOP

✓ There are two options for implementing the MOP.

- Integrate it with Meta objects. Every Meta object provides those functions of the MOP that operate on it.
- Implement the MOP as a separate component.

✓ Robustness is a major concern when implementing the MOP. Errors in change specifications should be detected wherever possible.

7. Define the base level

✓ Implement the functional core and user interface of the system according to the analysis model developed in step 1.

✓ Use Meta objects to keep the base level extensible and adaptable. Connect every base level component with Meta objects that provide system information on which they depend, such as type information etc.

✓ Provide base level components with functions for maintaining the relationships with their associated Meta objects. The MOP must be able to modify every relationship b/w base level and Meta level.

Example resolved:

Unlike languages like CLOS or Smalltalk. C++ does not support reflection very well-only the standard class type_info provides reflective capabilities: we can identify and compare types. One solution for providing extended type information is to include a special step in the compilation process. In this, we collect type information from the source files of the application, generate code for instantiating the' metaobjects, and link this code with the application. Similarly, the 'object creator' metaobject is generated. Users specify code for instantiating an 'empty' object of every type, and the toolkit generates the code for the metaobject. Some parts of the system are compiler-dependent, such as offset and size calculation.

Variants:

♥ *Reflection with several Meta levels*

Sometimes metaobjects depend on each other. Such a software system has an infinite number of meta levels in which each meta level is controlled by a higher one, and where each meta level has its own metaobject protocol. In practice, most existing reflective software comprises only one or two meta levels.

Known uses:

❖ **CLOS.**

This is the classic example of a reflective programming language [Kee89]. In CLOS, operations defined for objects are called generic functions, and their processing is referred to as generic function invocation. Generic function invocation is divided into three phases:

- ♣ The system first determines the methods that are applicable to a given invocation.
- ♣ It then sorts the applicable methods in decreasing order of precedence.
- ♣ The system finally sequences the execution of the list of applicable methods.

❖ **MIP**

It is a run-time type information system for C++. The functionality of MIP is separated into four layers:

- ♣ The first layer includes information and functionality that allows software to identify and compare types.
- ♣ The second layer provides more detailed information about the type system of an application.
- ♣ The third layer provides information about relative addresses of data members, and offers functions for creating 'empty' objects of user-defined types.
- ♣ The fourth layer provides full type information, such as that about friends of a class, protection of data members, or argument and return types of function members.

❖ **PGen**

It allows an application to store and read arbitrary C++ object structures.

- **NEDIS**
  NEDIS includes a meta level called run-time data dictionary. It provides the following services and system information:
  - Properties for certain attributes of classes, such as their allowed value ranges.
  - Functions for checking attribute values against their required properties.
  - Default values for attributes of classes, used to initialize new objects.
  - Functions specifying the behavior of the system in the event of errors
  - Country-specific functionality, for example for tax calculation.
  - Information about the 'look and feel' of the software, such as the layout of input masks or the language to be used in the user interface.
- **OLE 2.0**
  It provides functionality for exposing and accessing type information about OLE objects and their interfaces.

Consequences:

The reflection architecture provides the following *Benefits:*

- **No explicit modification of source code:**
- **Changing a software system is easy**
  MOP provides a safe and uniform mechanism for changing s/w. it hides all specific techniques such as use of visitors, factories from user.
- **Support for many kind of change:**
  Because Meta objects can encapsulate every aspect of system behavior, state and structure.

The reflection architecture also has *Liabilities:*

- **Modifications at meta level may cause damage:**
  - ✓ Incorrect modifications from users cause serious damage to the s/w or its environment. Ex: changing a database schema without suspending the execution of objects in the applications that use it or passing code to the MOP that includes semantic errors
  - ✓ Robustness of MOP is therefore of great importance.
- **Increased number of components:**
  It includes more Meta objects than base level components.
- **Lower efficiency:**
  Slower than non reflective systems because of complex relnp b/w base and meta level.
- **Not all possible changes to the software are supported**
  Ex: changes or extensions to base level code.
- **Not all languages support reflection**
  Difficult to implement in C ++

UNIT 6 – QUESTION BANK

No.	QUESTION	YEAR	MARKS
1	Explain the benefits and liabilities of microkernel pattern	Dec 09	10
2	Enumerate the implementation steps of reflection pattern	Dec 09	10
3	What are the steps involved in implementing the microkernel system?	June 10	12
4	What are the benefits and liabilities of reflection architecture pattern?	June 10	8
5	List and explain the participating components of a microkernel pattern	Dec 10	10
6	Explain the known uses of reflection pattern	Dec 10	10
7	Discuss the benefits and liabilities of microkernel pattern	June 11	10
8	Give the detailed explanation on the different known applications offered by the reflection pattern	June 11	10
9	Explain in brief, the components comprising the structure of microkernel architectural pattern	Dec 11	10
10	With an example, explain when the reflection architectural pattern is used. What are its benefits?	Dec 11	10
11	What are the steps involved in implementing the microkernel system?	June 12	8
12	Explain the known uses of reflection pattern	June 12	6
13	Explain the advantages and disadvantages of reflection architectural pattern	June 12	6