University of M’sila Faculty of Mathematics and Computer

Science Department of Computer Science

HCI Architecture Models

Academic Year 2024-2025

Prepared and presented by SAOUDI Lalia

1. Introduction

Architecture models define the software organisation of an interactive system.

- **Basic Principle**

- the separation between the functional core that implements concepts specific to a particular application domain and the interface that presents these concepts to the user and allows them to manipulate them.

- **Role**

- In theory, it should allow the interface to be modified without affecting the functional core and vice versa.

- It allows for the structuring of an interactive system and thus provides better modularity (which is crucial given the iterative method of building interfaces. It facilitates the software reuse of interactive system components, its maintenance, and its evolution).

1. Introduction

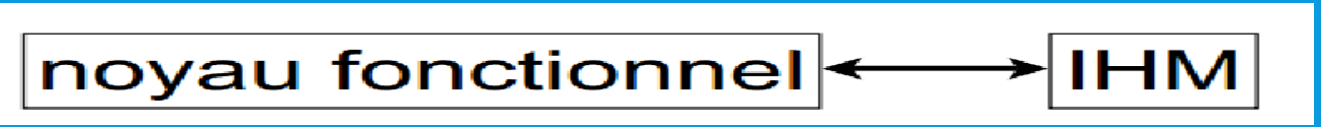
Garlan and Perry emphasise that the use of architectural models can have beneficial effects on at least five aspects of software development:

1. Understanding is simplified by presenting the structure of large systems at a suitable level of abstraction.
2. Reusability is encouraged
3. Evolution and maintenance greatly benefit from the definition of software architectures. By understanding the ramifications and connections of each component.
4. The analysis of systems is simplified
5. The management of industrial projects must rely on this essential element.

IHM and architecture

Base model

- Found in all models
- Separation of the interface and the application



Other architecture models

› Seeheim model or language model

› Object model

› MVC

› Multi-agent model: PAC

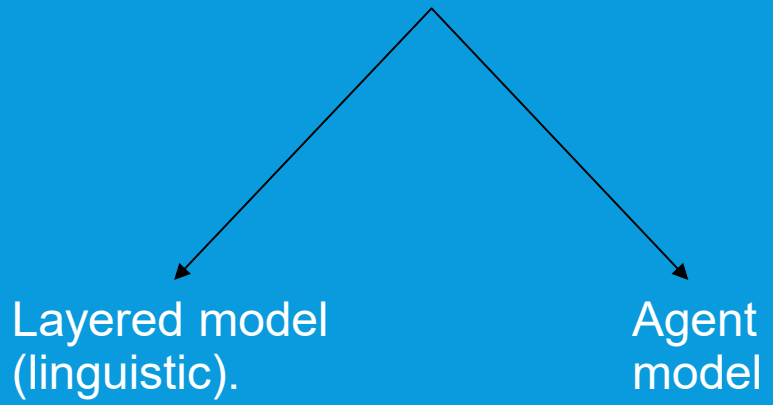
Definition: Software architecture

- Architecture models assume that an interactive system consists of a user interface and a functional core that relates to the application domain.
- Most models identify at least three types of elements.
  - Elements in direct contact with the user (presentations).
  - Elements in direct contact with the functional core or that are part of it (functional core interfaces, abstractions, etc.).
  - Elements for communication between the first two elements (controllers, adapters).

Software architecture models.

A view focused on the internal organisation and the division of software into modules. It describes:

- the nature of the different modules of a software
- the nature of the relationships between the modules.



Software architecture models: Layered model

- Layered models describe the overall structure of an interactive application in the form of logical layers. They are based on a linguistic approach to interaction, inspired by compiler architectures.
- They identify three aspects in interaction:

1)Lexical aspects:

refer to everything that can be assimilated to an input or output vocabulary

2) Syntactic aspects:

they can refer to input grammars representing the sequences of valid actions, or the spatial and temporal aspects of the display.

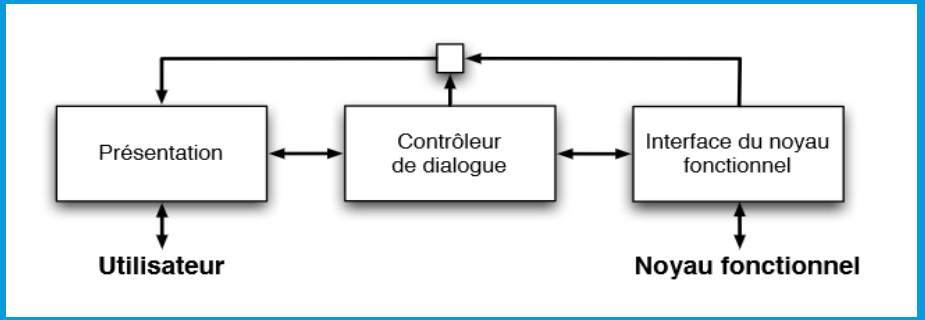
3) Semantic aspects:

correspond to the functional part of the application, which ultimately determines the meaning of an action and generates errors.

- **Main models: the Seeheim model, the model**

Layered model - Seeheim model

- The first reference architectural model for interactive applications was proposed at a workshop on interactive systems held in Seeheim (Germany) in 1985.
- It distinguishes three logical components:
  - **Presentation: layer that manages inputs and outputs**
  - **Dialogue Controller: manages the sequencing of inputs and outputs**
  - **Functional core interface: Application Interface Model**



Layered model - Seeheim model

1. The presentation component

• the actual production of outputs (display, sound production…) from the information received from the dialogue controller,
• the reception and processing (at a first level) of input events caused by the user's actions to then relay the resulting information to the dialogue controller.

2. The dialogue controller

- Where the structure of the dialogue between the user and the application is determined.
- The dialogue controller is responsible for sequencing operations at the user action level and for calls to domain functions
- It acts as a bridge between the concrete interface objects (presentation component) and the domain concepts (functional core).

Layered model - Seeheim model

3. The interface of the functional core

This component serves as a gateway between the functional core and the dialogue controller. It adjusts the differences between the formalisms used by these two components.

It generally contains a description of the domain entities that are directly linked to the interface as well as a description of the functional core procedures that can be invoked by the dialogue controller.

Example : ATM Machine System

1. Presentation Layer

Hardware: Screen, keypad, card reader, cash dispenser

Software:

Draws screens: "Enter PIN", "Select Transaction"

Captures keystrokes from keypad

Displays "Processing..." animation

Physically dispenses cash

Example : ATM Machine System

1. Presentation Layer

```
pseudocode// Example presentation pseudo-code
if (user_presses_key("5"))
   send_to_dialogue_control("KEY_5");

if (command_from_dialogue == "SHOW_BALANCE_SCREEN")
   display_balance_screen();
```

Example : ATM Machine System

2. Dialogue Control Layer

Manages workflow:

Card inserted → Ask for PIN

Valid PIN → Show menu

User selects "Withdraw" → Ask for amount

Valid amount → Check funds → Proceed

Enforces rules:

"User must enter PIN before seeing menu"

"Withdrawal amount must be multiple of 20"

"Maximum 3 incorrect PIN attempts"

Example : ATM Machine System

2. Dialogue Control Layer

```
State: AWAITING_PIN

  OnReceive("PIN_ENTERED"):

    if (validate_pin(pin))

      change_state(SHOW_MENU)

      send_to_presentation("DISPLAY_MENU")

    else

      attempts++

      if (attempts >= 3)          reject_card()
```

Example : ATM Machine System

3. Application Interface Layer (Functional Core)

Banking system backend:

Verifies PIN against database

Checks account balance

Processes withdrawal transaction

Updates database records

Contains business rules:

Daily withdrawal limits

Account status checks

Transaction logging

Example : ATM Machine System

Example: Withdraw $20000 Dz

1.User presses "20000" button

Presentation: Captures button press, sends "WITHDRAW_20000" to Dialogue Control

2. Dialogue Control processes request

Checks if amount is valid (multiple of 20) ✓

Checks workflow state (user authenticated, account selected) ✓

Sends request to Application: check_funds(account, 20000)

Example : ATM Machine System

3. Application performs business logic

- Queries database for balance

- Checks daily limit

- Returns: SUFFICIENT_FUNDS

4. Dialogue Control continues

- Sends commands to Presentation:

- show_processing_screen()

- dispense_cash(2000)

- print_receipt()

- Logs transaction complete

5. Presentation executes

- Physical cash dispensing
- Screen updates
- Receipt printing

Key Advantages of Seeheim Model

- ✅ Separation of Concerns: UI designers vs. application developers
- ✅ Reusability: Same application core with different UIs
- ✅ Maintainability: Change UI without touching business logic
- ✅ Multiple Platforms: Different presentations for same core

Limitations

- ❌ Dialogue Control can become complex for rich interactions
- ❌ Synchronous model - not ideal for event-driven modern UIs
- ❌ Too rigid for highly dynamic applications
- ❌ Feedback loops between layers can be inefficient

Layered model - Seeheim model

Example 2:

A company wishes to have a pizza ordering system. With this system, the user:

- selects by choosing the pizza(s) he wishes to order.
  - For example, he might want to choose a specific type of pizza (e.g. Standard, Queen, etc.), a specific size (e.g. Large, Medium, Small), and specific toppings (e.g. Cream, Cheese).
- At any time, he can also view his order list : the number of pizza(s) ordered, as well as the total price of his order.
- Finally, a 'Order' button allows him to actually place his order.

Example: Module split according to the Seeheim architecture

according to the Seeheim software architecture model. To do this, you need to divide the code into four distinct modules:

1. The Application module: this module implements the minimal functional core of the application. This module is represented by three classes:

- a class "Pizza" that encapsulates the concept of pizza and allows for the calculation of its price.
- A class "Order" that manages the list of pizzas. This is implemented using a "Vector" containing "Pizza" objects.
- A class "Client",

Example: Modular breakdown according to the Seeheim architecture.

2. The Application Interface module: this module implements the methods allowing the Dialog Controller to access the features of the Application module. It is implemented as a single class and has numerous methods for accessing information about the order and performing actions on this order. These methods will notably allow:

- to calculate the price of a pizza
- to add a pizza to the order,
- to delete the order,
- to remove the last pizza added,
- to calculate the price of the order,
- to obtain the entire order (in the form of a string).

- Of course, this module makes use of the Pizza and Order modules to obtain this information.

Example: Modularisation according to the Seeheim architecture

3. The Dialogue Control Module: this module manages the human-machine dialogue. It is this module that takes the initiative to trigger, for instance, the update of the order list based on the **data provided by the Presentation** and to request that this Presentation displays the result. This module must, in no circumstances, make use of objects from the graphical toolbox, nor from the functional kernel. It must do so through the Presentation or the Application Interface. It therefore has methods that are called by the Presentation following the user's actions:

- the user makes a selection with one of the radio buttons: requestChoice()
- the user presses the 'Add Pizza' button: requestSelection()
- the user presses the 'Clear Last' button: requestClearLast()
- the user presses the 'Clear List' button: requestClearList()
- the user presses the 'Order' button: requestOrder()
- the user wants to exit the application: RequestExit() ;

Example: Module breakdown according to the Seeheim architecture

4. The Presentation module: this module manages the graphical display and the retrieval of events triggered by the user.

It also provides the Dialogue Controller with methods to access and modify data entered by the user or displayed to the user:

reading the characteristics of the pizza: getTypePizza(), getsizePizza(), getSupplementCream() and getSupplementCheeze() ,getNbPizza(…)

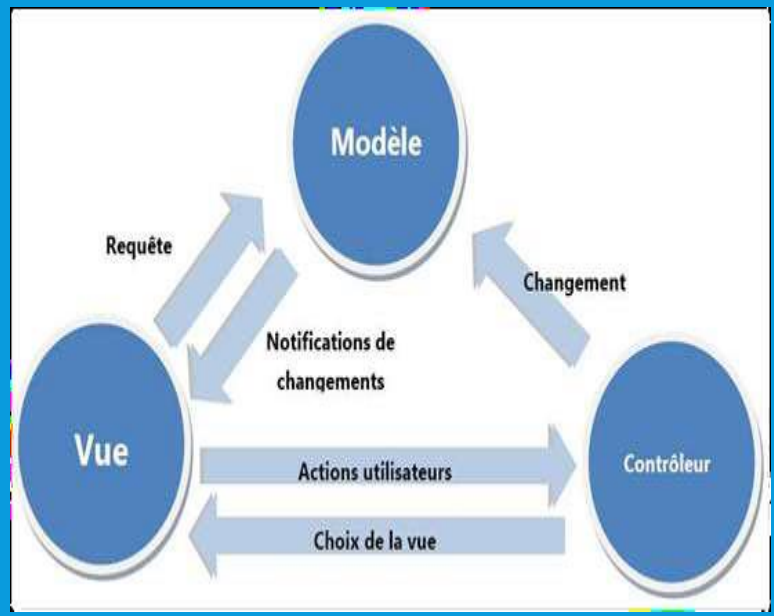
writing the price of the chosen pizza, the list of orders, the number of pizzas, the total price and the conclusion of the order: setPricePizza(…), setOrderList( ), , setTotalPrice(…) and showOrder()

The MVC (Model-View-Controller) Model

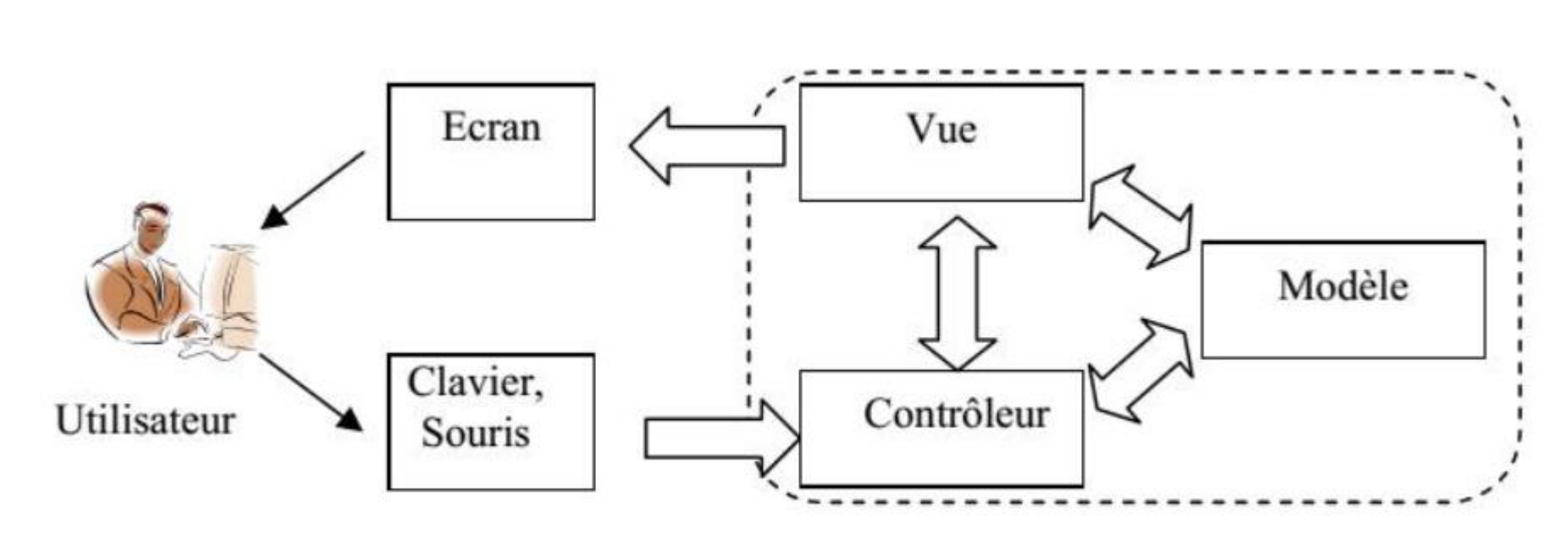
MVC is: A software architecture (a way to structure an application or a set of software)
In this architecture, three types of objects collaborate:

- The Model is the semantic object that we want to visualise and modify.
- The View is the graphical object, responsible for the visualisation and graphical editing of a facet of the model.
- The controller handles event management to synchronise and
- update the view or the model.

  It does not perform any processing; it analyses the client's request and simply calls the appropriate model and returns the view corresponding to the request.



The agent model MVC (Model-View-Controller)



A SIMPLE EXAMPLE OF MVC IN PHP

- We will create a simple web application that allows for managing a task list (To-Do List) according to the MVC architecture:
- 1. Folder structure

- mvc-php/
- /app/
- /controllers/
- TaskController.php
- /models/
- Task.php
- /views/
- taskList.php
- /public/
- index.php
- /config/
- database.php

A SIMPLE EXAMPLE OF MVC IN PHP

- The model (Task.php)
- The Task.php model handles access to the database data. It includes the necessary methods to add, delete, and retrieve tasks.

1.THE MODEL LAYER (TASK .PHP)

```php
<?php

class Task {

private $db;

public function __construct() {

$this->db = new PDO('mysql:host=localhost;dbname=mvc_example', 'root', '');

}

public function getAllTasks() {

$query = $this->db->prepare("SELECT * FROM tasks");

$query->execute();

return $query->fetchAll(PDO::FETCH_ASSOC);

}

public function addTask($taskName) {

$query = $this->db->prepare("INSERT INTO tasks (name) VALUES (:name)");

$query->bindParam(':name', $taskName);

$query->execute();

}

public function deleteTask($taskId) {

$query = $this->db->prepare("DELETE FROM tasks WHERE id = :id");

$query->bindParam(':id', $taskId);

$query->execute();

}

}
```

2. TaskController.php

The TaskController.php controller manages interactions between the model and the view. It receives requests from the client, processes the data, and determines which view to display.

The controller includes methods to

display the list of tasks (index()),

add a task (addTask()), and

delete a task (deleteTask()).

These methods interact with the model to access data and with the view to display the results.

2. TaskController.php

```php
<?php
require_once '../app/models/Task.php';
class TaskController {
private $taskModel;
public function __construct() {
$this->taskModel = new Task();
}
public function index() {
$tasks = $this->taskModel->getAllTasks();
require '../app/views/taskList.php';
}
```

```php
public function addTask($taskName) {

$this->taskModel->addTask($taskName);

header('Location: /');

}

public function deleteTask($taskId) {

$this->taskModel->deleteTask($taskId);

header('Location: /');

}

}
```

3. The View (taskList.php)

The view taskList.php is responsible for displaying data to users. It receives data from the controller and presents it in a readable format.
The view uses a foreach loop to display each task retrieved by the controller. It also includes a form to add new tasks.

3. The View (taskList.php)

```
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Task List</title>
</head>
<body>
<h1>Task List</h1>
<ul>
<?php foreach ($tasks as $task): ?>
<li><?php echo htmlspecialchars($task['name']); ?>
<a href="/delete?id=<?php echo $task['id']; ?>">Delete</a>
</li>
<?php endforeach; ?>
</ul>

<form method="POST" action="/add">

<input type="text" name="taskName" placeholder="New task">

<button type="submit">Add</button>
```

4. The Router (index.php)

The router is the part of the application that determines which controller and method to execute based on the received request.

```
<?php
require_once '../app/controllers/TaskController.php';
$controller = new TaskController();
if ($_SERVER['REQUEST_METHOD'] === 'POST' && isset($_POST['taskName'])) {
$controller->addTask($_POST['taskName']);
} elseif (isset($_GET['id'])) {
$controller->deleteTask($_GET['id']);
} else {
$controller->index();
}
```

The router analyses the HTTP method (GET or POST) to determine the action to be performed.

If it receives a POST request with a task name, it calls addTask().

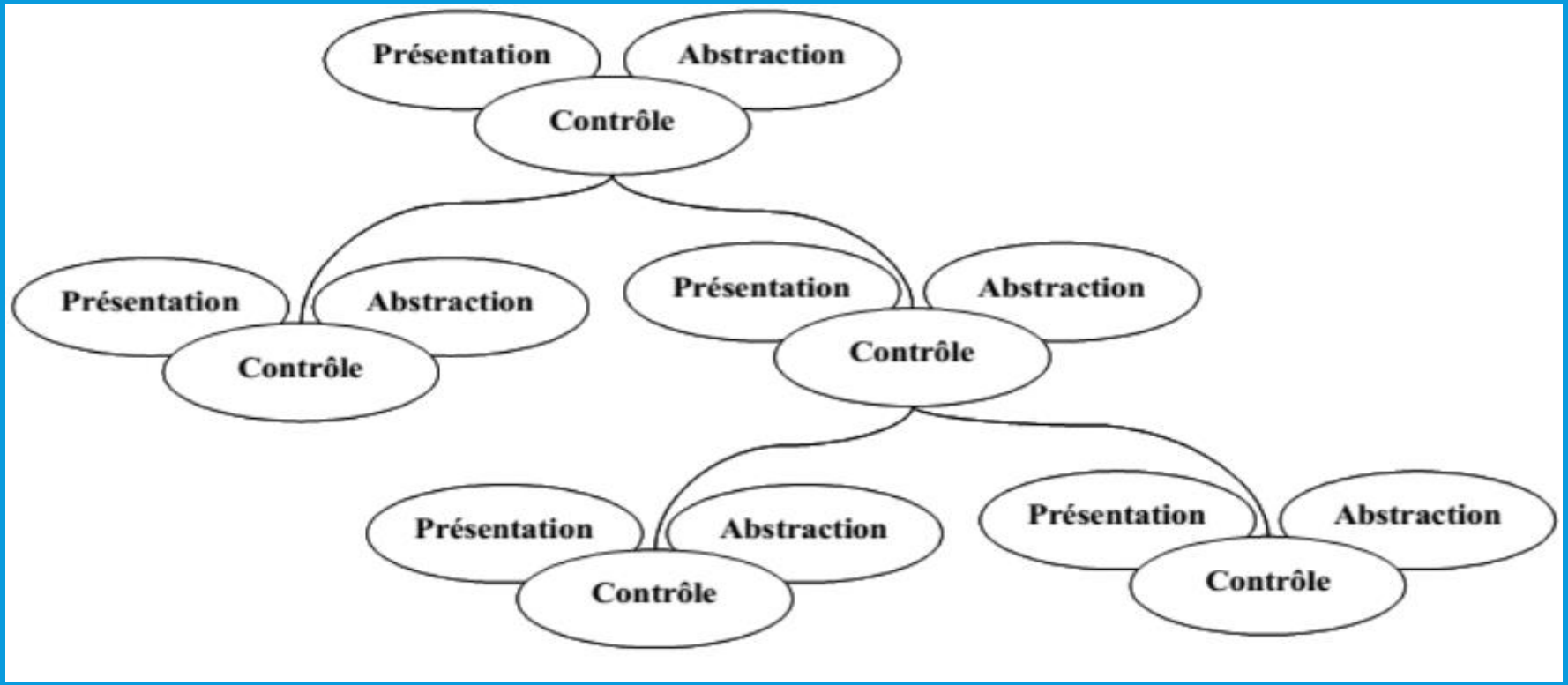
If it receives a GET request with a task ID, it calls deleteTask(). By default, it calls the index() method to display the list of tasks.

The agent model PAC (Presentation-Abstraction-Control)

The PAC model decomposes the interactive application into a hierarchy of interactive agents structured in three facets:

- **The Presentation component is responsible for the transmission (visual, auditory, etc.) of information to the user and for acquiring the user's input.**
- **The Abstraction component contains the data and semantic processes.**
- **The Control component is the engine of the dialogues that occur between the agent and the user, and between the agent and other agents.**

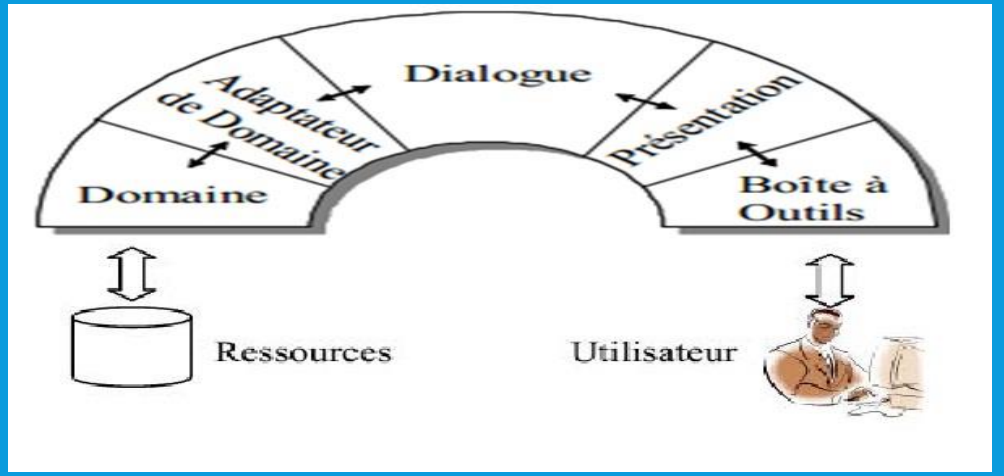
The agent model PAC (Presentation-Abstraction-Control)



Structure of an interactive PAC application

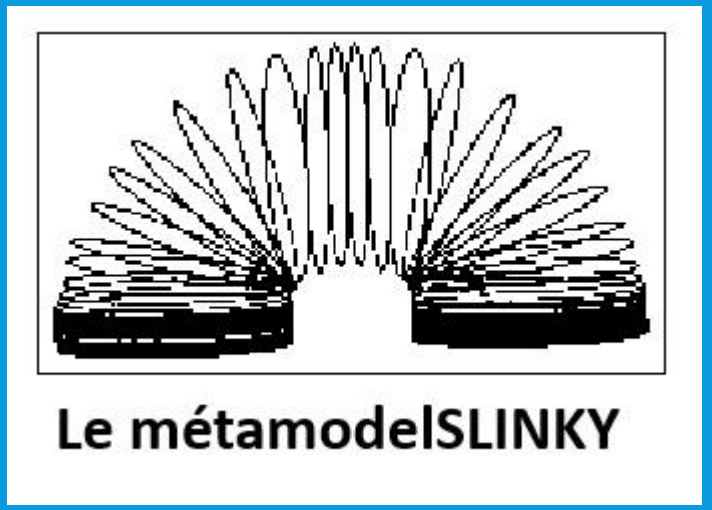
The ARCH model

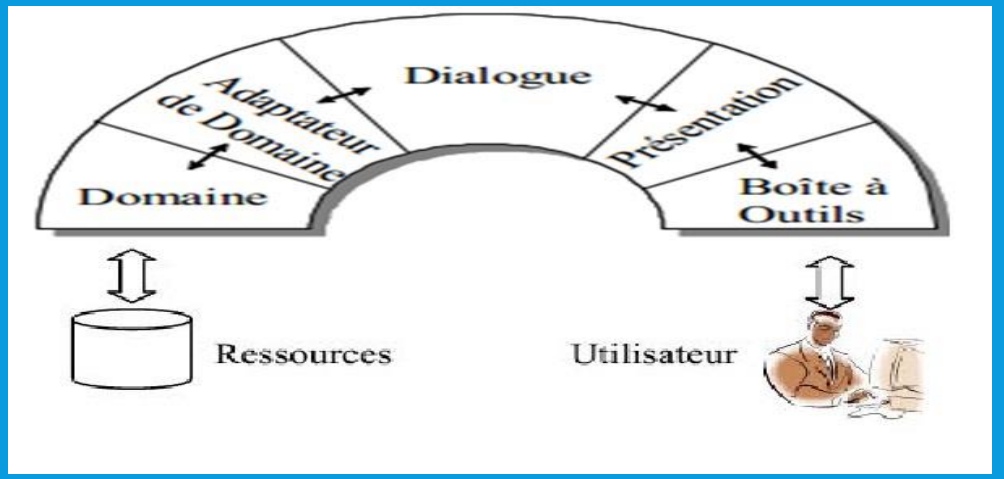
- Software architecture pattern introduced in 1992 to structure interactive software.
- Refines the Seeheim model by taking greater inspiration from current graphical toolkits
- Identifies five components in interactive systems.
- is based on the conceptual components of the Seeheim model by adding two intermediate components.
- It takes the Seeheim dialogue controller as the centre of the arch and refines the decomposition of the other two pillars.



The ARCH model

- The functional core and the interaction component (usually a toolkit) form the legs of the arch, as in most cases these two components exist before the development of the interface itself and thus provide the starting point.
- The presentation and the adapter of the functional core enable adaptation between the three other components.
- These adaptations are not always necessary: the Slinky meta-model allows for their disappearance or their merging into the other elements of the arch.





The agent model

- **Agent models: close to object-oriented programming languages and modern direct manipulation interfaces.**
- Ferber defined the agent as a computational entity, either real or abstract, independent with a limited lifespan, capable of acting upon itself and its environment.
- An agent communicates with its peers and its behaviour is the result of its observations, its knowledge, and its interactions with other agents.