OOP - Introduzione

Introduzione alla Programmazione Orientata agli Oggetti

Andrea Colleoni

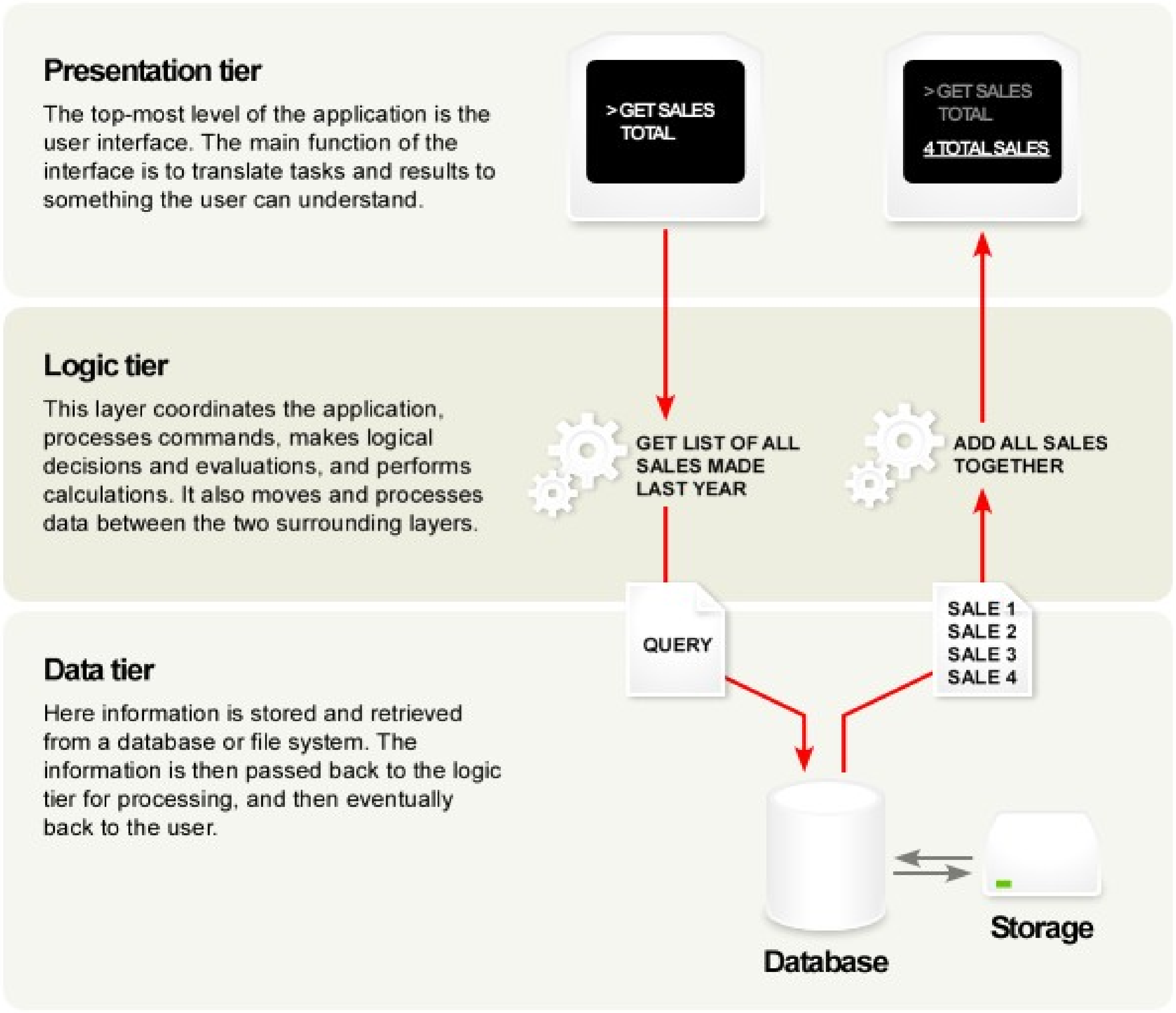
Lezione 4 – Architectural patterns

Introduzione ai Pattern Architetturali

Fondamenti e motivazioni...

- I pattern architetturali hanno un ambito più ampio rispetto ai pattern di progetto (design pattern)
- Similmente a quanto già osservato per i design pattern, questi pattern forniscono una gamma di soluzioni comunemente ritenute accettabili per i sistemi di elaborazione
- I pattern architetturali sono riconosciuti da molto più tempo

Multitier ...



Presentation tier

The top-most level of the application is the user interface. The main function of the interface is to translate tasks and results to something the user can understand.

Logic tier

This layer coordinates the application, processes commands, makes logical decisions and evaluations, and performs calculations. It also moves and processes data between the two surrounding layers.

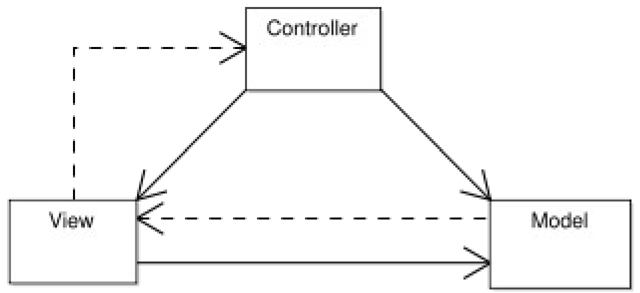
Data tier

Here information is stored and retrieved from a database or file system. The information is then passed back to the logic tier for processing, and then eventually back to the user.

...Multitier...

- In software engineering, multi-tier architecture (often referred to as n-tier architecture) is a client-server architecture, originally designed by Jonathon Bolster of Hematites Corp, in which an application is executed by more than one distinct software agent. For example, an application that uses middleware to service data requests between a user and a database employs multi-tier architecture. The most widespread use of "multi-tier architecture" refers to three-tier architecture.

Model-View-Controller...



- It is an architectural pattern used in software engineering. In complex computer applications that present a large amount of data to the user, a developer often wishes to separate data (model) and user interface (view) concerns, so that changes to the user interface will not affect data handling, and that the data can be reorganized without changing the user interface. The model-view-controller solves this problem by decoupling data access and business logic from data presentation and user interaction, by introducing an intermediate component: the controller.

...Model-View-Controller...

- Model
  - The domain-specific representation of the information that the application operates. Domain logic adds meaning to raw data (e.g., calculating whether today is the user's birthday, or the totals, taxes, and shipping charges for shopping cart items). Many applications use a persistent storage mechanism (such as a database) to store data. MVC does not specifically mention the data access layer because it is understood to be underneath or encapsulated by the Model

...Model-View-Controller...

- View
  - Renders the model into a form suitable for interaction, typically a user interface element. Multiple views can exist for a single model for different purposes.
- Controller
  - Processes and responds to events, typically user actions, and may invoke changes on the model.

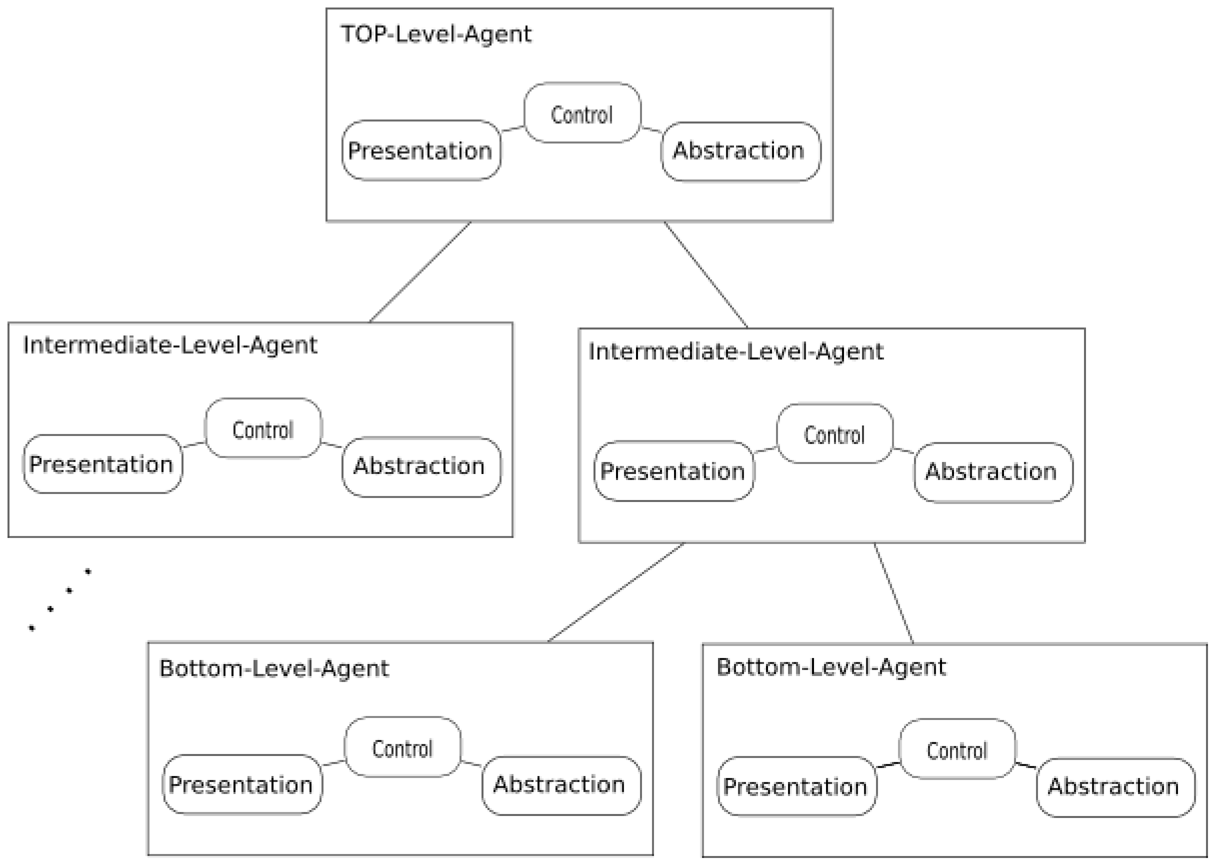
MVC: esempi...

- .NET: Windows Forms
  - In WinForms, a .NET framework, the patterns for the view and controller are well defined. The model is left to the developer to design.
    - Model
      - Just like ASP.Net, WinForms does not strictly require a model. The developer has the option to create a model class, but may choose to forget it and have the event handlers in the controller perform any calculations and data persistence. Again, using a model to encapsulate business rules and database access is both possible and preferable. It is left to developers to design the Model.
    - View
      - A class inheriting from either Form or Control handles the responsibilities of the view. In the case of WinForms, the View and Controller are compiled into the same class. This differs from ASP.Net, which uses inheritance, and Smalltalk, which have separate classes with pointers to one another
    - Controller
      - The duties of the controller are split among three places. The generation and passing of events starts at the OS level. Inside the .Net framework, the Form and Control classes route the event to the proper event handler. The event handlers typically reside in the Form or Control class and are implemented as delegates for the user interface events (e.g., button click, form load, listbox selection changed).

...MVC: esempi...

- .NET: ASP.NET
  - Page controller
    - Model
      - DataSets and DataTables are the most common use of the model in .NET projects. A typed DataSet allows one to create an entity-specific model based on a database.
    - View
      - The ASPX and ASCX files generally handle the responsibilities of the view, although it can also come from compiled server controls. With this pattern, the view object may choose to inherit from the controller object. This is different from the Smalltalk implementation, in which separate classes have pointers to one another.
  - A more flexible option is to have the code-behind classes of the ASPX and ASCX to implement an Interface for a View, which can then be passed to the Controller. With this approach, the Controller is loosely coupled to and largely unaware of the actual concrete implementer of the View interface, be it an ASPX or an ASCX.
- Controllers
  - The duties of the controller are split between two places. The generation and passing of events is part of the framework and more specifically the Page and Control classes. The handling of events is usually done in the code-behind class. However, moving code specific to the transition between views in a separate Controller is a good practice. In turn, it becomes possible to centralize the registration of Observers in the isolated Controller.

Presentation-abstraction-control...



...Presentation-abstraction-control...

- Presentation-abstraction-control
  - (PAC) is a software architectural pattern, somewhat similar to model-view-controller (MVC). PAC is used as a hierarchical structure of agents, each consisting of a triad of presentation, abstraction and control parts. The agents (or triads) communicate with each other only through the control part of each triad. It also differs from MVC in that within each triad, it completely insulates the presentation (view in MVC) and the abstraction (model in MVC), this provides the option to separately multithread the model and view which can give the user experience of very short program start times, as the user interface (presentation) can be shown before the abstraction has fully initialized.

...Presentation-abstraction-control

- Hierarchical-Model-View-Controller (HMVC)
  - A subset or variation of PAC under the name Hierarchical-Model-View-Controller (HMVC) was published in an article[1] in JavaWorld Magazine, the authors apparently unaware[2] of PAC which was published 13 years earlier. The main difference between HMVC and PAC is that HMVC is less strict in that it allows the view and model of each agent to communicate directly, thus bypassing the controller.

Service Oriented Architecture (SOA)

- It is an architectural style that guides all aspects of creating and using business processes, packaged as services, throughout their lifecycle, as well as defining and provisioning the IT infrastructure that allows different applications to exchange data and participate in business processes regardless of the operating systems or programming languages underlying those applications. SOA represents a model in which functionality is decomposed into small, distinct units (services), which can be distributed over a network and can be combined together and reused to create business applications. These services communicate with each other by passing data from one service to another, or by coordinating an activity between one or more services. It is often seen as an evolution of distributed computing and modular programming.

SOA

- Distributed computing
  - segmented or parallel computing
  - Different environments (Oss, FileSystems, etc.)
- Consider: RPC, CORBA, DCOM, RMI
- EDI
- WebServices
  - WSDL, UDDI and SOAP

SOA: principi...

- Reuse, granularity, modularity, composability, componentization, and interoperability
- Compliance to standards (both common and industry-specific)
- Services identification and categorization, provisioning and delivery, and monitoring and tracking

...SOA: principi...

- **Service Encapsulation:** A lot of existing web-services are consolidated to be used under the SOA Architecture. Many a times, such services have not been planned to be under SOA.
- **Service Loose coupling:** Services maintain a relationship that minimizes dependencies and only requires that they maintain an awareness of each other
- **Service contract:** Services adhere to a communications agreement, as defined collectively by one or more service description documents
- **Service abstraction:** Beyond what is described in the service contract, services hide logic from the outside world
- **Service reusability:** Logic is divided into services with the intention of promoting reuse
- **Service composability:** Collections of services can be coordinated and assembled to form composite services
- **Service autonomy:** Services have control over the logic they encapsulate
- **Service optimization:** All else equal, high-quality services are generally considered preferable to low-quality ones
- **Service discoverability:** Services are designed to be outwardly descriptive so that they can be found and assessed via available discovery mechanisms

Implicit Invocation

- The idea behind implicit invocation is that instead of invoking a procedure directly, a component can announce (or broadcast) one or more events. Other components in the system can register an interest in an event by associating a procedure with the event. When the event is announced the system itself invokes all of the procedures that have been registered for the event. Thus an event announcement implicitly causes the invocation of procedures in other modules."
- Implicit invocation is used by some authors for a style of software architecture in which a system is structured around event handling, using a form of callback. It is closely related to Inversion of control and what is known informally as the Hollywood Principle.