National School of Geographic Sciences

Geomatics and Information Systems Technology

Architectural Patterns

ex MapReduce

Author:
Mohamed Amjad
LASRI

Supervisor:
Emmanuel Bardiere

December 4, 2014

Abstract

Architectural patterns provide reusable solutions for common architectural problems. Patterns usually don't contains code that you can cut and past; instead, they contain architectural and design information that you build into your solution. Nowadays we define architectural patterns sub-domains adapted, more or less, to each type of situations. In this documentary research project we presente what architectural patterns are, How are they integrated into an information system project? We also provide

Acknowledgements

Thanks God, thanks Mum!

License

Copyright (C) 2014 Mohamed Amjad LASRI. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

1 Introduction

Architectural patterns are commonly recognised as efficient solutions to a reccuring problem in the software architecture field. This type of patterns is generally implemented to solve some problems related to hardware performance, business risks, big data analysis, servers high availability ...and others. nevertheless, "Architectural patterns" is a confusing expression, the reader must distinguish between this expression and "software architecture styles". For example, SOA is a software architecture style, but ESB -which is an architectural pattern- is a manner to implemente a SOA. In the schema diagram bellow we presente some software architecture style and the related architectural patterns. The reader must also distinguish between two others confusing concepts: design patterns and architectural patterns: Design patterns are usually associated with code level commonalities. It provides various schemes for refining and building smaller subsystems. It is usually

influenced by programming language. Some patterns pale into insignificance due to language paradigms. Design patterns are medium-scale tactics that flesh out some of the structure and behavior of entities and their relationships.

While architectural patterns are seen as commonality at higher level than design patterns. Architectural patterns are high-level strategies that concerns large-scale components, the global properties and mechanisms of a system.

2 Architectural Patterns

In this section we list some basic architectural styles and the patterns that help you design them. There may be more than one pattern for each style. Both MVC and PAC, for example, are in the interactive system style.

2.1 From Mud to Structure

From Mud to Structure is the root and entry point to our pattern language. Its featured patterns help to transform the mud of requirements and constraints we usually start with into a coarse-grained software structure with clearly separated, tangible parts that make up the system being developed, and address several key concerns of sustainable software architectures: operational aspects such as performance and availability, as well as developmental qualities like extensibility and maintainability.

2.1.1 Layered Architecture

an example of a "From Mud to Structure" pattern is the layered architecture. This architectural pattern have been since the beginning of digital computers — or at least since the early 1960s. Modern hardware technology and languages accentuate the usefulness of layered architectures. The International Standards Organization (ISO) Open Systems Interconnection (OSI) seven-layer model (see the Figure bellow) facilitates communication between computers. The model consists of two separate but parallel stacks of layers; each layer provides a higher level of functionality than the layer below it. Within the two stacks, the layer N in one stack is a peer of the layer N in the other stack. Logically, communication is between the peer layers in the two stacks; actually, only the bottommost layer directly communicates between the two stacks.

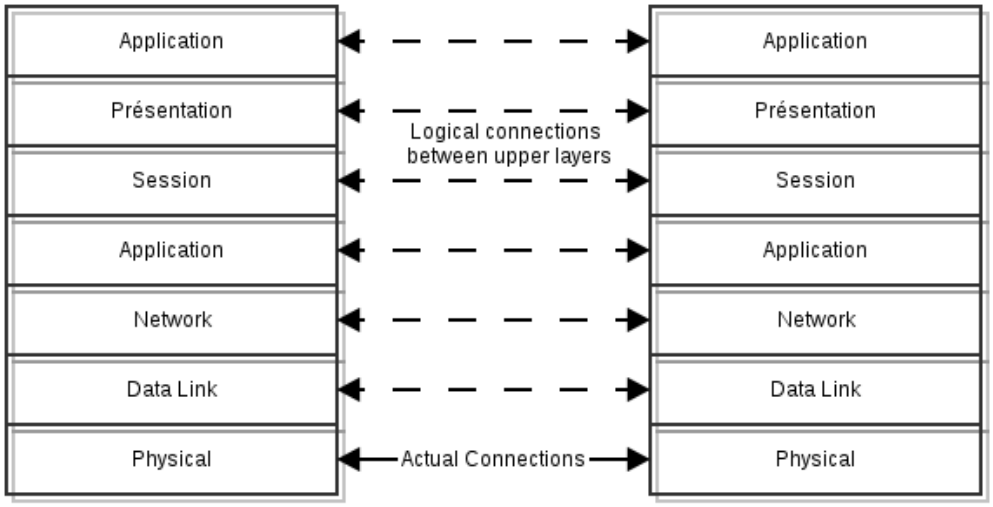

Figure 1: the OSI architectural pattern

Layers can be different sizes, hosting different numbers of protocols. Over time, in fact, the communications-stack diagram has evolved into many variations. Some layers have many alternatives — such as SIP, FTP, Telnet, and HTTP in layer 7 (the Application layer) — and other layers have few alternatives. Something that's true of layered architectures in general is true of the OSI model as well: Changes to a layer affect only that one layer. In other words, the communication protocol chosen at a lower layer doesn't affect the higher- level functionality provided at its higher level. You're free to pick and choose the protocols to use in each layer — just keep in mind that for peers to talk to each other, the peer layers in the two stacks must use the same protocol.

2.2 Distributed systems

A distributed system consists of a collection of autonomous nodes (computers or servers), connected through a network and distribution middleware, which enables computers to coordinate their activities and to share the resources of the system, so that users perceive the system as a single, integrated computing facility. The most known architectural patterns that implemente the distributed systems architecture:

- 2-tiers (client-server), 3-tiers, n-tiers;
- Peer-to-Peer;
- Broker;
- Serivce Oriented;

2.2.1 Broker

A message broker is an architectural pattern for message validation, message transformation and message routing. It mediates communication amongst applications, minimizing the mutual awareness that applications should have of each other in order to be able to exchange messages, effectively implementing decoupling.

The purpose of a broker is to take incoming messages from applications and perform some action on them. The following are examples of actions that might be taken in the broker:

- Route messages to one or more of many destinations;
- Transform messages to an alternative representation;
- Perform message aggregation, decomposing messages into multiple messages and sending them to their destination, then recomposing the responses into one message to return to the user;
- Interact with an external repository to augment a message or store it;
- Invoke Web services to retrieve data;
- Respond to events or errors;
- Provide content and topic-based message routing using the publish–subscribe pattern.

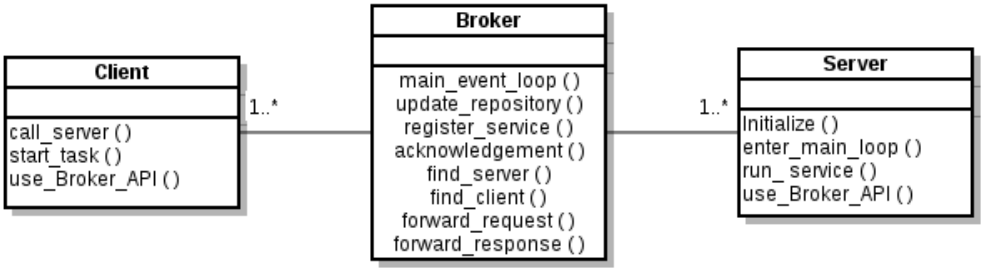

Figure 2: A typical example of the Broker pattern

Apache ActiveMQ as an implementation of the Broker pattern Apache ActiveMQ is an open source message broker written in Java together with a full Java Message Service (JMS) client. It provides "Enterprise Features" which in this case means fostering the communication from more than one client or server. Supported clients include Java via JMS 1.1 as well as several other "cross language" clients. The communication is managed with features such as computer clustering and ability to use any database as a JMS persistence provider besides virtual memory, cache, and journal persistency.

2.2.2 Service Oriented Architecture

Service-oriented architecture (SOA) is a design pattern based on distinct pieces of software providing application functionality as services to other applications via a protocol. This is known as service-orientation. It is independent of any vendor, product or technology.

A service is a self-contained unit of functionality, such as retrieving an online bank statement. Services can be combined by other software applications to provide the complete functionality of a large software application. SOA makes it easy for computers connected over a network to cooperate. Every computer can run an arbitrary number of services, and each service is built in a way that ensures that the service can exchange information with any other service in the network without human interaction and without the need to make changes to the underlying program itself.

There are no industry standards relating to the exact composition of a service-oriented architecture, although many industry sources have published their own principles.

2.3 Interactive Systems

2.3.1 Model-View-Controller

Model–view–controller (MVC) is a software architectural pattern for implementing user interfaces. It divides a given software application into three interconnected parts, so as to separate internal representations of information from the ways that information is presented to or accepted from the user.

Components: The central component of MVC, the model, captures the behavior of the application in terms of its problem domain, independent of the user interface. The model directly manages the data, logic and rules of the application. A view can be any output representation of information, such as a chart or a diagram; multiple views of the same information are possible, such as a bar chart for management and a tabular view for accountants. The third part, the controller, accepts input and converts it to commands for the model or view.

Interactions: In addition to dividing the application into three kinds of components, the model–view–controller design defines the interactions between them.

- A controller can send commands to the model to update the model's state (e.g., editing a document). It can also send commands to its associated view to change the view's presentation of the model (e.g., by scrolling through a document).
- A model notifies its associated views and controllers when there has been a change in its state. This notification allows the views to produce updated output, and the controllers to change the available set of commands. In some cases an MVC implementation might instead be "passive," so that other components must poll the model for updates rather than being notified.
- A view requests information from the model that it uses to generate an output representation to the user.

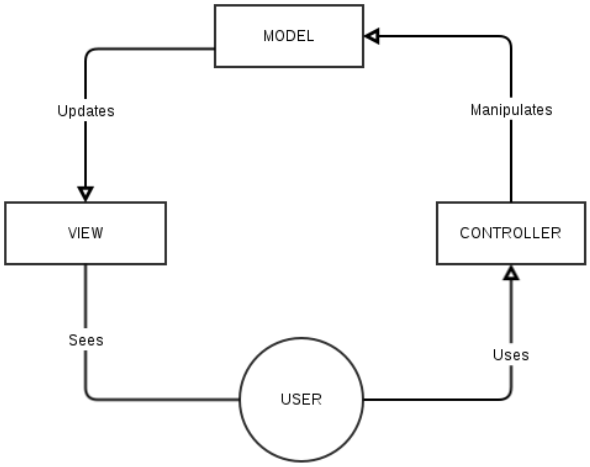

Figure 3: MVC architecture

3 MapReduce

MapReduce is a programming model designed for processing large volumes of data in parallel by dividing the work into a set of independent tasks. MapReduce programs are written in a particular style influenced by functional programming constructs, specifically idioms for processing lists of data. In this section we explain the nature of this programming model and how it can be used to write programs which run in the Hadoop environment[1].

3.1 MapReduce Basics:

3.1.1 Functional Programming:

all data elements in MapReduce are immutable[2], meaning that they cannot be updated. If in a mapping task you change an input (key, value) pair, it does not get reflected back in the input files; communication occurs only by generating new output (key, value) pairs which are then forwarded by the Hadoop system into the next phase of execution.

[1]The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. it implementes MapReduce

[2]an immutable object is an object whose state cannot be modified after it is created

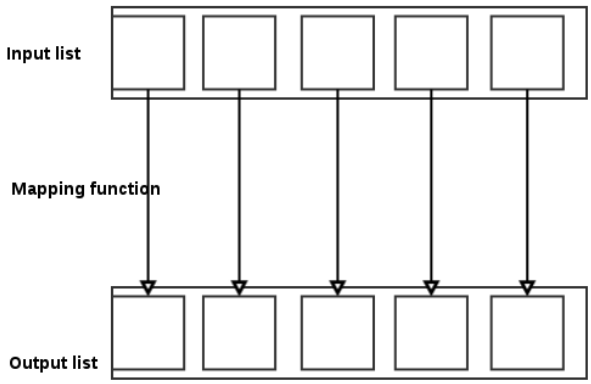

Figure 4: MapReduce List processing

3.1.2 List processing

Conceptually, MapReduce programs transform lists of input data elements into lists of output data elements. A MapReduce program will do this twice, using two different list processing idioms: map, and reduce. These terms are taken from several list processing languages such as LISP, Scheme, or ML.

As an example of the utility of map: Suppose you had a function toUpper(str) which returns an uppercase version of its input string. You could use this function with map to turn a list of strings into a list of uppercase strings. Note that we are not modifying the input string here: we are returning a new string that will form part of a new output list.

3.1.3 Mapping lists

The first phase of a MapReduce program is called mapping. A list of data elements are provided, one at a time, to a function called the Mapper, which transforms each element individually to an output data element.

As an example of the utility of map: Suppose you had a function toUpper(str) which returns an uppercase version of its input string. You could use this function with map to turn a list of strings into a list of uppercase strings. Note that we are not modifying the input string here: we are returning a new string that will form part of a new output list.

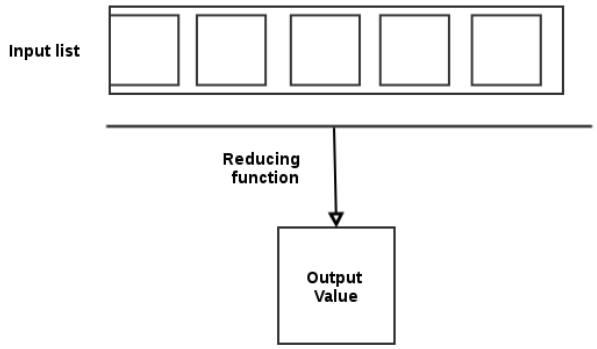

Figure 5: MapReduce: reducing lists

3.1.4 Reducing lists

Reducing lets you aggregate values together. A reducer function receives an iterator of input values from an input list. It then combines these values together, returning a single output value.

Reducing is often used to produce "summary" data, turning a large volume of data into a smaller summary of itself. For example, "+" can be used as a reducing function, to return the sum of a list of input values.

3.2 Sections

Use section and subsection commands to organize your document. LaTeX handles all the formatting and numbering automatically. Use ref and label commands for cross-references.

3.3 Comments

Comments can be added to the margins of the document using the todo command, as shown in the example on the right. You can also add inline comments too:

This is an inline comment.

Here's a comment in the margin!

Figure 6: This is a figure caption.

3.4 Tables and Figures

Use the table and tabular commands for basic tables — see Table **??**, for example. You can upload a figure (JPEG, PNG or PDF) using the files menu. To include it in your document, use the includegraphics command as in the code for Figure 6 below.

3.5 Mathematics

LaTeX is great at typesetting mathematics. Let $X_1, X_2, \ldots, X_n$ be a sequence of independent and identically distributed random variables with $\mathrm{E}[X_i] = \mu$ and $\mathrm{Var}[X_i] = \sigma^2 < \infty$, and let

$$S_n = \frac{X_1 + X_2 + \cdots + X_n}{n} = \frac{1}{n}\sum_{i}^{n} X_i$$

denote their mean. Then as n approaches infinity, the random variables $\sqrt{n}(S_n - \mu)$ converge in distribution to a normal $\mathcal{N}(0, \sigma^2)$.

3.6 Lists

You can make lists with automatic numbering ...

1. Like this,
2. and like this.

. . . or bullet points . . .

- Like this,
- and like this.

We hope you find writeLaTeX useful, and please let us know if you have any feedback using the help menu above.