Lecture 32 — Software Architecture Patterns

Jeff Zarnett
jzarnett@uwaterloo.ca

Department of Electrical and Computer Engineering
University of Waterloo

December 29, 2014

Software Architecture Pattern

Software Architecture Pattern: a general, re-usable solution for how to structure software.

At a high level, nearly all software has three major elements:

1. Logic
2. Data
3. User Interface

Software Architecture Patterns

Different patterns apply to different problems.

Large software almost never has exactly 1 architecture pattern.

Different patterns used in different parts of the same program.

Some deviation from the pattern is normal.

Architecture Patterns: Motivation

Similar to why we use life cycle models.

In university assignments, you can get away without structure.
    This tends to result in the “monolith” pattern.

Bigger software will require deliberate choices of architecture.
    Otherwise, a mess to clean up later.

Architecture Patterns: Monolith

The monolith is a single, self-contained program.

No defined modules, no clear lines between different parts.

Logic, user interface, and data all mixed together.

Probably your university work looks like this.

Architecture Patterns: Monolith

Mainframe applications used to be written like this.

Complexity spirals out of control.

Eventually becomes unmaintainable.

Architecture Patterns: Monolith

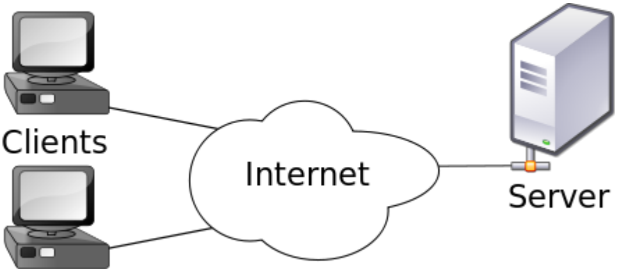
Often faster to write than other structures.

Tradeoff: more difficult to understand and maintain.

Changes might have unexpected consequences.

Not necessarily the wrong choice: see Microsoft Word.

Architecture Patterns: Client-Server



`http://en.wikipedia.org/wiki/File:Client-server-model.svg`

Two roles: client and server.

Architecture Patterns: Client-Server

Client and Server communicate via a request-response pattern.

Client sends requests; server sends responses.

The two must understand each other. Protocol.

Examples: E-mail, World Wide Web, Banking

Architecture Patterns: Client-Server

There is a clear line dividing client and server.

Most often, logic and data are on the server; client handles UI.

Often client & server on different physical machines.
  But this is not a requirement.

Architecture Patterns: Client-Server

Example: opening a web page, `www.google.com`.

Browser sends request to Google. Google server responds.

The browser does not know or care how the server generates its answer.

The server does not know or care how the request was generated, or how the browser draws the response page.

Architecture Patterns: Client-Server

Historical Note: Mainframes and Batch Jobs

At one time, lots of computing used client-server.

PCs were expensive and mainframes were common.

Desk computers were “thin clients” or “dumb terminals”;
- Little or no processing power of their own.
- Sent work to the server and displayed responses.

Architecture Patterns: Client-Server

Mainframes and Batch Job Example

When programming, the programmer edits code on her PC.

To compile, submit a *batch job* – server request.

Server takes the code, compiles it, and returns the result.

Architecture Patterns: Client-Server

Batch Job processing is uncommon now.

PC prices fell and their power increased (starting in the 80s).

More efficient to buy powerful PCs; have them do the work.

Compiling is now done on local machines.
  Although there are often official build servers.

Architecture Patterns: Client-Server

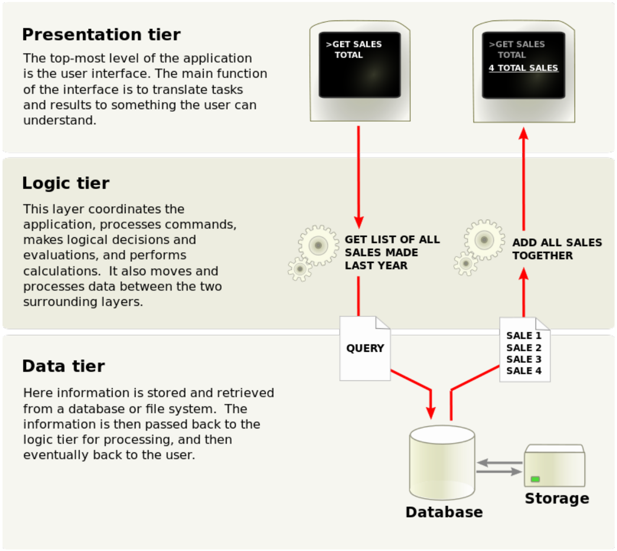
Typically possible to replace either the client or server
...as long as the communication protocol is unchanged.

Example: we can access Google with Firefox or Chrome.

Advantage over monolith: separated concerns.

Risk: central server crash might stop the whole system!

Architecture Patterns: Three-Tier



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.

http://en.wikipedia.org/wiki/File:Overview_of_a_three-tier_application_vectorVersion.svg

Architecture Patterns: Three-Tier

Consists of three distinct but interacting parts.

The user interface, logic, and data reside in their own areas.

These can be separate physical systems, but don’t have to be.

Architecture Patterns: Three-Tier

Each tier can be changed without affecting the other parts.

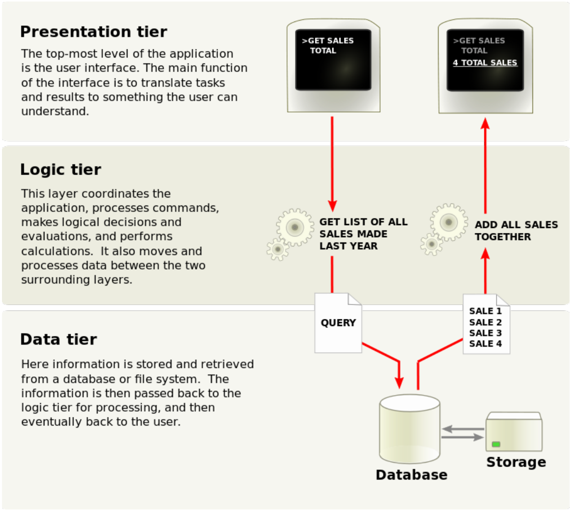
Example: present data on Android and not just Windows.

Example 2: change database from mySQL to Oracle.

Tiers can be subdivided further, especially the logic tier.
Then we have an *n-Tier* architecture.

Architecture Patterns: Three-Tier

Let's examine the tier descriptions more closely.



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.

http://en.wikipedia.org/wiki/File:Overview_of_a_three-tier_application_vectorVersion.svg

Architecture Patterns: Three-Tier

Can be used in combination with client-server model.

Most common split: client has UI, server has logic and data.

Note that the client can have logic of its own
    So the dividing line could be anywhere in the logic tier.

Presentation tier does not communicate with the data tier.
    Communication only between adjacent tiers.

Architecture Patterns: Three-Tier

Client-Server interaction can occur between any *adjacent* tiers.

Obvious case: presentation tier a client to logic tier's server.

From the perspective of the data tier: data tier is the server, responding to requests from a client, the logic tier.

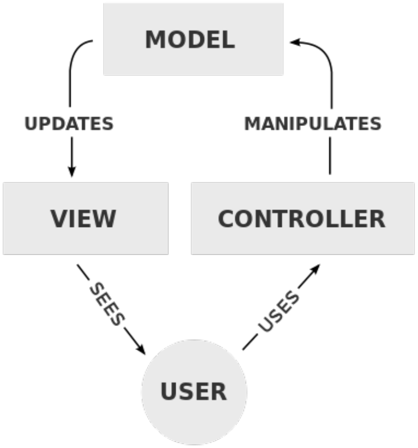
Architecture Patterns: Three-Tier

Revisit the Google example. This time we'll search!

The browser remains the presentation tier in this example.

1. Browser sends to the server our search request.
2. The server analyzes our request and queries the database for matches on these search terms.
3. The database returns the results to the logic tier.
4. The logic tier formats the results into a webpage.
5. The logic tier sends the answer back to your browser.
6. The browser draws the page on the screen.

Architecture Patterns: Model-View-Controller



http://en.wikipedia.org/wiki/File:MVC-Process.svg

Architecture Patterns: Model-View-Controller

Model-View-Controller is often abbreviated MVC.

Software is divided into the model, view, and controller.

The pattern also defines the interactions of these three.

Architecture Patterns: Model-View-Controller

Model-View-Controller is often abbreviated MVC.

Software is divided into the model, view, and controller.

The pattern also defines the interactions of these three.

Example: `Contact` object, holding contact info for a person or business (e.g., phone number)

Architecture Patterns: Model-View-Controller

The Model:

- Holds a reference to the `Contact`
- Contains business logic and functions.
- Notifies view(s) and controller(s) of changes to `Contact`
- View, Controller must use model to access `Contact`

Architecture Patterns: Model-View-Controller

The View:

- Any representation of `Contact` data.
- We can have multiple views of the same data.
- Requests from the model info it needs to display.

Architecture Patterns: Model-View-Controller

The Controller:

- Accepts input; converts it to commands.
- Commands sent to model manipulate data.
- Commands sent to view change the view (e.g. scrolling).

Architecture Patterns: Model-View-Controller

MVC Example.

User wants to set phone number to null.

Clicks on button “Clear Phone Number”.

The click is input to the controller.

Controller turns this into a command for the model:

```
setPhoneNumber(null);
```

Architecture Patterns: Model-View-Controller

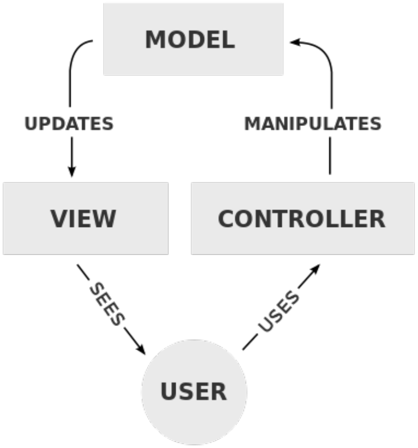
MVC Example Continued.

Model executes command - `Contact` is updated.

Model notifies the view that the `Contact` has changed.

View re-draws the screen to reflect the change.

Architecture Patterns: Model-View-Controller



`http://en.wikipedia.org/wiki/File:MVC-Process.svg`

Architecture Patterns: Model-View-Controller

Considered good practice because it separates logic from UI.

We can have different ways of viewing the data.

One element (often the view) can be changed without affecting others.

Architecture Patterns: Model-View-Controller

MVC overlaps with client-server.

The client might call up a record from the server and then use MVC for the user to manipulate it,

One of the model actions is save changes
(send them back to the server for storage).

Thin-client system: the client has the view and controller; server has the model.

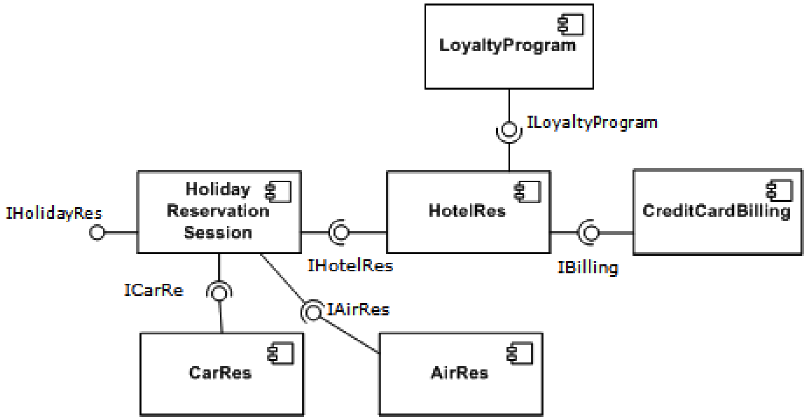
Architecture Patterns: Model-View-Controller

MVC overlaps with three tier as well.

The view and controller are presentation tier
model in the logic tier.

MVC, Three-Tier, and Client-Server can all work together.

Architecture Patterns: Modular Architecture



http://en.wikipedia.org/wiki/File:Component-based-Software-Engineering-example2.gif

Architecture Patterns: Modular Architecture

As defined so far, the client and server (or tiers) might themselves be described as being monolithic.

One or both of those might be too complex for monolith.

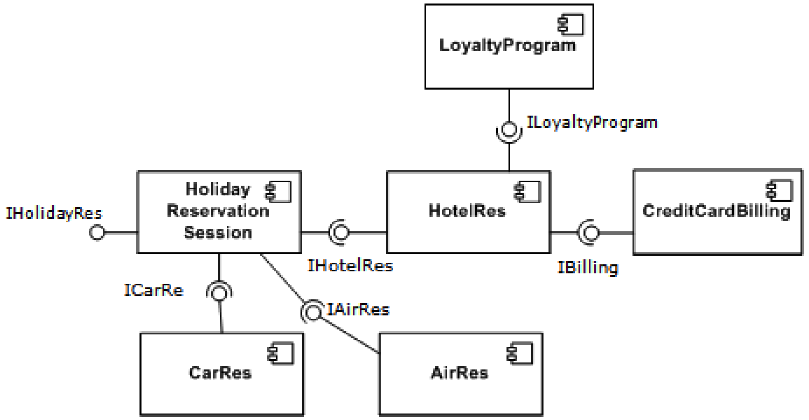
Solution: modules!

The final software product is comprised of a number of modules which interact.

A module, sometimes called a component, contains a related set of methods/functions or data.

The modular architecture is also sometimes called "Component-Based Software Engineering".

Architecture Patterns: Modular Architecture



http://en.wikipedia.org/wiki/File:Component-based-Software-Engineering-example2.gif

Architecture Patterns: Modular Architecture

The distinct boxes indicate that parts can be swapped in and out.

If the loyalty program moves to a new airline, swap `AirRes` without changing anything else.

Under ideal circumstances, components can be re-used.