**Large Scale Software Configurations: Concepts and Styles**

Ric Holt, 22 Aug 2012 CS746

This note assumes that the structure/organization of large software systems can be reasonably divided into these three somewhat overlapping categories as follows:

1. **Classical** software architecture (as described by Garlan and Shaw, commonly designed to run on a single hardware system),
2. **Empirical** architecture (as described in Brown and Wilson, commonly developed using open source methods), and
3. **Large scale software configurations** (large numbers of precessors, often connected via internet).

This note discusses these three, concentrating on the third: large scale software systems. References will be made to sources of definition/discussion using internet links.

**1. LAYERS AND PLATFORMS**

**Layers of Abstraction.** In some layered systems, each layer introduces new abstraction, e.g., a DBMS introduces relational tables with corresponding operations. In some cases the abstraction if robust, meaning that users cannot in a direct way, access to implementation of the abstraction. For example, Unix (as a layer) introduced the abstraction of files as sequences of bytes. All three categories use such layers.

**API** (interface for programming): Interface (for source code)).

**Software platform**: Includes a [hardware architecture](http://en.wikipedia.org/wiki/Hardware_architecture) and a [software framework](http://en.wikipedia.org/wiki/Software_framework) (including [application frameworks](http://en.wikipedia.org/wiki/Application_framework)), where the combination allows [software](http://en.wikipedia.org/wiki/Computer_software), particularly [application software](http://en.wikipedia.org/wiki/Application_software), to run. <http://en.wikipedia.org/wiki/Computing_platform> Can be considered to be a layer of abstraction.

**Runtime system**: Is a [software](http://en.wikipedia.org/wiki/Software) component designed to support the execution of [computer programs](http://en.wikipedia.org/wiki/Computer_program) written in some [computer language](http://en.wikipedia.org/wiki/Computer_language), e.g., run-time support for Java. <http://en.wikipedia.org/wiki/Run-time_system>

**Framework**: Frameworks contain key distinguishing features that separate them from normal *libraries*, e.g., Eclipse IDE:

1. [**inversion of control**](http://en.wikipedia.org/wiki/Inversion_of_control) - In a framework, unlike in libraries or normal user applications, the overall program's [flow of control](http://en.wikipedia.org/wiki/Control_flow) is not dictated by the caller, but by the framework.[[1]](http://en.wikipedia.org/wiki/Software_framework#cite_note-0)
2. **default behavior** - A framework has a default behavior. This default behavior must actually be some useful behavior and not a series of [no-ops](http://en.wikipedia.org/wiki/NOP).
3. [**extensibility**](http://en.wikipedia.org/wiki/Extensibility) - A framework can be extended by the user usually by selective overriding or specialized by user code providing specific functionality. [plug-ints]
4. **non-modifiable framework code** - The framework code, in general, is not allowed to be modified. Users can extend the framework, but not modify its code.

<http://en.wikipedia.org/wiki/Framework>

**Plug-in:** Is a set of [software components](http://en.wikipedia.org/wiki/Software_component) that adds specific abilities to a larger [software application](http://en.wikipedia.org/wiki/Software_application). If supported, plug-ins enable customizing the functionality of an application. For example, plug-ins are commonly used in [web browsers](http://en.wikipedia.org/wiki/Web_browser) to play video, scan for viruses, and display new file types. <http://en.wikipedia.org/wiki/Plug-in_%28computing%29>

**2. HIGH LEVEL SOFTWARE CONFIGURATIONS**

**Middleware**: Is [computer](http://en.wikipedia.org/wiki/Computer) [software](http://en.wikipedia.org/wiki/Software) that provides services to [software applications](http://en.wikipedia.org/wiki/Software_application) beyond those available from the [operating system](http://en.wikipedia.org/wiki/Operating_system)… The term is most commonly used for software that enables communication and management of data in [distributed applications](http://en.wikipedia.org/wiki/Distributed_application). In this more specific sense *middleware* can be described as “the dash in '[client-server](http://en.wikipedia.org/wiki/Client-server)'”. <http://en.wikipedia.org/wiki/Middleware>

**Client server:**    Thick / thin client
    Ajax: Thick client model

 <http://en.wikipedia.org/wiki/Client%E2%80%93server_model>



**Gorton’s model of middleware**: [Gorton, Essential Software Architecture]

Business Process Orchestrators

Message Brokers

Application Servers

Transport

*Message-Oriented Middleware,*

*Distributed Objects Systems*

*J2EE, CCM, .NET*

*BizTalk, WebSphere Message Broker,*

*SonicMQ*

*BizTalk, TIBCO StaffWare,*

*ActiveBPEL*

**J2EE**: Known as *Java 2 Platform, Enterprise Edition* or *J2EE* until the name was changed to *Java EE* in version 5. The current version is called *Java EE 6*

**SOA: Service Oriented Architecture**: Is a set of principles and [methodologies](http://en.wikipedia.org/wiki/Methodologies) for designing and developing [software](http://en.wikipedia.org/wiki/Software) in the form of [interoperable](http://en.wikipedia.org/wiki/Interoperability) [services](http://en.wikipedia.org/wiki/Service_%28systems_architecture%29). These services are well-defined [business](http://en.wikipedia.org/wiki/Service-oriented_architecture) [functionalities](http://en.wikipedia.org/wiki/Function_%28computer_science%29) that are built as [software components](http://en.wikipedia.org/wiki/Software_component) (discrete [pieces of code](http://en.wikipedia.org/wiki/Modular_programming) and/or [data structures](http://en.wikipedia.org/wiki/Data_structure)) that can be [reused](http://en.wikipedia.org/wiki/Code_reuse) for different purposes… [Service-orientation](http://en.wikipedia.org/wiki/Service-orientation) requires [*loose coupling*](http://en.wikipedia.org/wiki/Loose_coupling) of services with [operating systems](http://en.wikipedia.org/wiki/Operating_system) and other technologies that underlie applications. SOA separates functions into distinct units, or services,[[1]](http://en.wikipedia.org/wiki/Service-oriented_architecture#cite_note-Bell-0) which developers make accessible over a network in order to allow users to combine and reuse them in the production of applications. <http://en.wikipedia.org/wiki/Service-oriented_architecture>

**Software ecosystem**: Is “a collection of software systems, which are developed and co-evolve in the same environment” <http://en.wikipedia.org/wiki/Software_ecosystem>

**Cloud computing:** Is the use of [computing](http://en.wikipedia.org/wiki/Computing) resources (hardware and software) that are delivered as a service over a [network](http://en.wikipedia.org/wiki/Computer_network) (typically the [Internet](http://en.wikipedia.org/wiki/Internet))… Cloud computing entrusts remote services with a user's data, software and computation. <http://en.wikipedia.org/wiki/Cloud_computing>

**Distributed computing frameworks**: Use of interconnections of hundreds to thousands of largely independent computers, e.g., Hadoop.

**3. LIBRARIES AND REPOSITORIES**

**Libraries and utilities**: Libraries provide commonly used software often divided into routines or classes. Utilities are commonly used software, e.g., a sorting utility, e.g., library of statistics calculations.

**Software repositories**: A collection of code (often source code), usually very large.
Software collections. Example: The collected releases of Linux source code over the last decade.

**Software collection**: A particular kind of software repository. Collected software packages, often evaluated for quality. Examples: Debian and Red Hat.

**4. DESIGN AND DEVELOPMENT**

**PitL** (Programming in the Large): Developing large programs, e.g., 1 MLOC. Development of product takes time measured in tens to hundreds of people-months

**PitM** (Programming in the Many): Programming with large number of developers, e.g., over 100.

**Scenarios, Use Cases**:  examples of how the system is used... includes high level view of data (state) and high level concept of control flow. Should illustrate the essential functionality of the system. Used to design and explain high level architecture.

**IEEE Std 1471**: <http://en.wikipedia.org/wiki/IEEE_1471> A formalization of the means of describing software architecture. Inspired by classical software architecture. Thought by some to be cumbersome and to not be practical