ExcaliburHistory

What's Excalibur? Where did the ideas for the project come from?

What is Avalon? What happened to Avalon?

Gaining an understanding of Excalibur's origins can help tremendously in understanding what Excalibur is today and where the project is headed. Excalibur grew out of the Apache Avalon project and that is where our story begins...

What is Avalon?

Avalon is a component framework for Java.

Ok, so what is a component framework?

Component Oriented Programming (COP) combines nifty design patterns such as Inversion of Control (IoC) and Separation of Concerns to traditional Object Oriented Programming. Design patterns are essentially 'best practices' for solving common programming problems. COP applies these practices to components – reusable, replacable, composable objects which when assembled together (usually inside of a container) make for a well designed, secure, and easily maintained application.

That sounds cool, but kinda abstract and generic...

In a way, yes. Avalon is somewhat abstract and generic, just like object oriented programming is abstract and generic. Avalon is a COP framework for Java. All the basic algorithms and utilities for using those design patterns are in place. There are even a couple of containers and prebuilt components available. You can then use Avalon to build just about anything, from a web application to a new web framework, from a desktop standalone app to a new application server. Most of the 'extras' of Avalon are designed to build server-like applications, but the framework itself can be used anywhere.

Avalon's Story

Once upon a time

Let's start off with some basic history. Avalon emerged from the Java Apache Server Framework before the days of Apache Jakarta. During development of the JServ project (which laid the foundation for Tomcat), many of the developers realized that the ideas being used at the time could be abstracted into a general application framework. This framework eventually became the Avalon Framework of today. The idea was to develop a generic component oriented framework from which many different types of containers and reusable components could be created.

In the beginning there was the **Avalon Framework**. It described the *contracts* or interfaces between various components (such as lifecycle methods) and provided some general utilities for using these interfaces. One could easily create an application which *only* used the framework. However, in doing so, one would often end up repeatedly writing a lot of component wiring code. In order to provide default implementations of this wiring code and some more advanced components and utilties, **Excalibur** was born.

Excalibur provided a set of basic components and utilities which made working with the framework much easier. One of these components was the **Excalibur Component Manager** or **ECM** which did all the work of getting all your component and configuration data sorted out and started. It was the first *containe* of sorts. ECM didn't have many advanced features, but it was simple to work with and could be used in any number of environments. For example, the XML publishing framework Cocoon used ECM internally.

With time new expectations and requested features meant that other Avalon containers were under development. Soon ECM would have company with Phoenix.

The Rise of the Phoenix

Phoenix was a complete standalone container for Avalon. The Phoenix container was designed as a *microkernel*. While it could be used for other sorts of applications, most development focused on server applications such as web servers, FTP servers, telnet servers, etc. The Apache mail server James used Phoenix as its container in this way. Phoenix applications would take a number of components and bundle them together in what was called a *block.* A block generally referred to a complete application, such as a database or FTP server, although you could have inter-block dependencies. Blocks would be packaged up with configuration and assembly files into a *.sar* archive, similar to the *.ear* files for J2EE applications. Phoenix would then launch all the SAR blocks contained within a particular startup directory.

Applications running within Phoenix used the Avalon Framework just as ECM components would. In fact, if you were careful to only depend on the framework for development, with a little work you could get applications written for ECM to run in Phoenix and visa versa.

Cornerstone became a repository of Phoenix blocks – larger components which could be dropped into Avalon Phoenix and provide services and resources to user developed components and applications. There was some overlap between components developed in Cornerstone and Excalibur, but in general, Cornerstone components were targeted for server side applications running in Phoenix.

ECM and Phoenix grew and stablized. However, as these stories go, a refactoring was on the horizon and changes were in the wind...

How Components Became Services

In the beginning there were only components. The components had a role defined by a java interface and an implementation defined by a concrete java class. In ECM roles and components could be described in a set of XML configuration files, generally one for the roles and one for the implementations. In Phoenix, roles were still roles and components were still components, but they were defined in xinfo files scattered across the various jar archives that would make up an application. This was done to allow developers to deploy a jar file that contained not only the interfaces and implementations, but also the basic meta-data. Thus, the xinfo files and the conf files had the same purpose but were used by different containers.

At this time, all components were children of the one `org.apache.framework.component.Component` interface. A brave developer scaled Mt. Doom and tossed the Component interface and all the other marker interfaces into the fiery pit, thus freeing all components from bondage of the one Component.

Upon returning from this quest, the developer said, "All Components shall now be dubbed Services" and a new set of Service Managers and Service Selectors appeared that could handle any Object, not just Components. These Service utilities performed the exact same functions as their deprecated Component counterparts, but didn't require everything be a Component. That is:

```
Component componentManager.lookup(String role);
```

became

```
Object serviceManager.lookup(String role);
```

In this sense, Components ARE Services. But now the Avalon community had two names for the same thing which inevitably caused confusion, especially for new users.

Fortress and Merlin arrive

Effort was made in Phoenix to support the new Service semantics, but instead of rewriting ECM a decision was made to create a new ECM-like container which could use Components and Services alike. Thus **Fortress** was born.

Fortress supports legacy ECM components but provides a number of features like meta-data configuration (instead of a "roles" file), dynamic service activation, lifecycle extensions, instrumentation support and so on. Like ECM, Fortress is also "embeddable" in that you can easily start up a Fortress container in your own application be it a Java Swing client or a Servlet. Fortress provides no default standalone client (i.e.- there's no "main" method class in Fortress) and doesn't do much classloader magic, making embedding a little more predictable. Fortress was released in the summer of 2003 and replaced ECM as Avalon's light weight container of choice.

While Fortress grew from ECM, **Merlin** grew from Phoenix, though it quickly developed beyond its roots. Merlin focused on a strict separation between container concerns and component concerns using a model driven approach. While it could also be used in an emedded mode, Merlin was especially suited for application servers. It provided a very rich (and somewhat complicated) set of deployment options such as hierachical containers, remote service and block loading, integration with Maven-style repositories and more.

Container Bonanza

Now Avalon was left with a host of containers: Phoenix, Merlin, Fortress and the deprecated ECM which Fortress was meant to replace. Merlin and Phoenix targetted application server development while Fortress and ECM targetted embedded container development. At first glance this didn't appear to be a problem though new users to Avalon were often confused by which container to use. However, the real difficulty lie in the fact that it was difficult to create a single component which worked in all the containers. They all used the Avalon framework, but each had their own configuration and meta-data format, custom "context" values, and differing service lookup semantics.

Consequently there was much discussion about the future of Avalon. Should Avalon become an "umbrella" project supporting all the containers? Should the containers be spun off into their own projects? Should we attempt to merge the containers into a single unified Avalon platform?

Avalon Divided

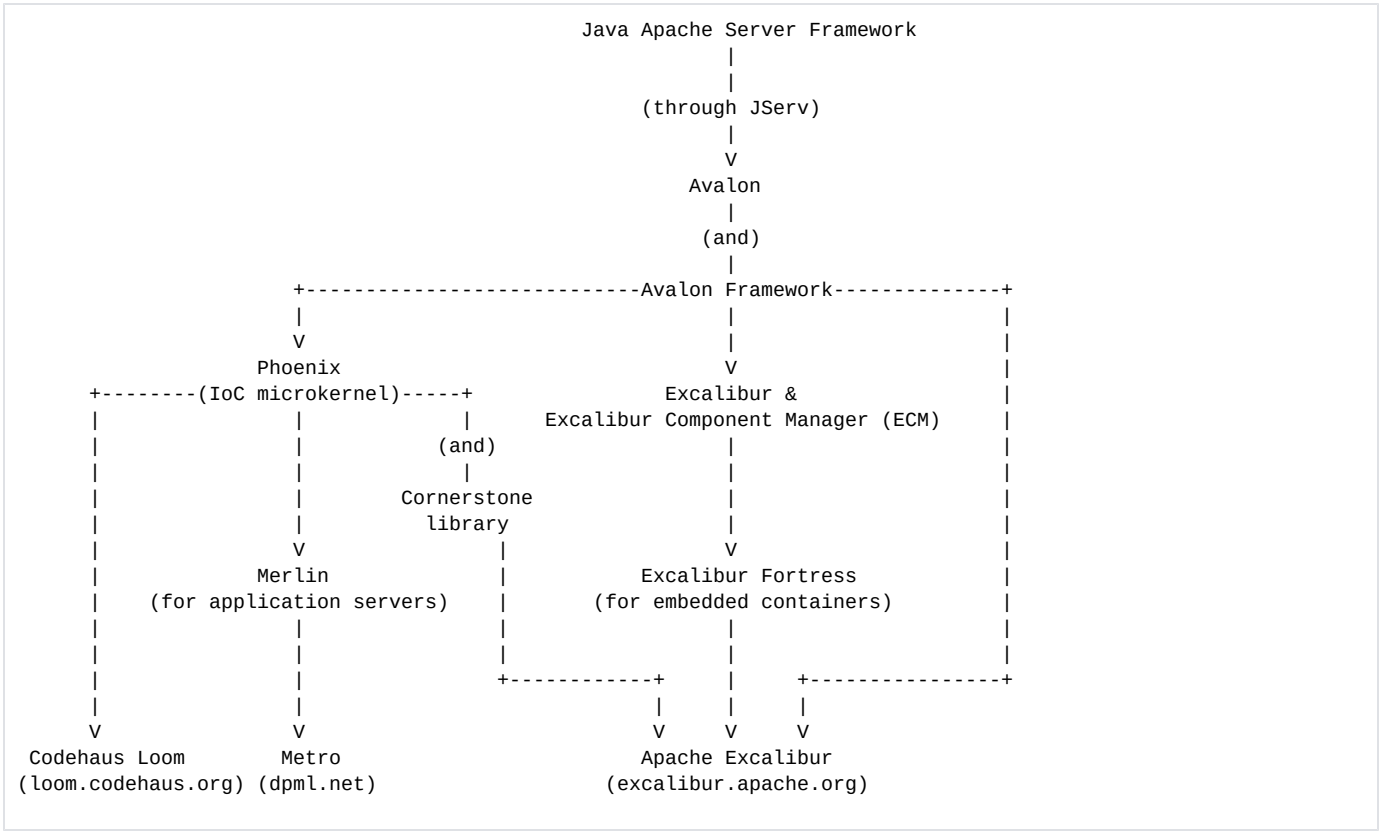
These questions were debated for a lengthy period and left a split amongst the developers. One was expelled and soon after forked the original Phoenix container into a new project at Codehaus called Loom. Following this, there was an effort to unify all Avalon development under the Merlin platform; however, this too resulted in conflict.

In early 2004 there were several attempts to reorganize Avalon. The first step was the start of the Apache Excalibur project in order to ensure support for ECM and Fortress users. Unfortunately, similar efforts to transform Merlin into its own top level Apache project failed and the developers began a new fork of Merlin called Metro as part of the DPML project outside of the ASF.

With the fork of Merlin, the final Avalon assets which included the original Avalon Framework and the Cornerstone library were transferred to Excalibur. Following this transfer, Avalon was closed in late 2004, its community having split into new projects at Apache and elsewhere.

Excalibur Today

Today Excalibur primarily serves to maintain and support existing ECM, Fortress, and Avalon Framework users. There has been some discussion and early efforts at new container and component development but these actions are still in the early stages.



The whole story as a diagram (this diagram was derived wholly from the description on this page and so may be quite incorrect)