ContainerStory

The Story of the Avalon Containers

A common question of users new to Avalon is, "Which container should I use?" or "What's the difference between ECM, Phoenix, Fortress, and/or Merlin?" Consider this the *verbose* answer. 🙂

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 we have today. The idea was to develop a generic component oriented framework from which many different types of containers and reusable components could be created. For a more detailed history of Avalon, see the community section on the main website.

So in the beginning there was the **Avalon Framework**. It basically just 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 simply *just* used the framework. However, in order to provide some more advanced components and utilties (which were not essential to the framework, but still useful) **Excalibur** was born.

Excalibur held 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 *container* or sorts. ECM didn't have a lot of "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.

Of course, 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 the first really complete full fledged standalone container for Avalon. Phoenix was not only a container, but a *microkernel*. While it could be used for other sorts of applications, most Phoenix development revolved around server applications such as web servers, FTP servers, telnet servers, etc. 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.

Thus Phoenix was a full application server of sorts. 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 return 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 converse with 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);

So in this sense, Components ARE Services. But now the Avalon community had two names for the same thing and this is generally were confusion arises. In *my* humble opinion, the word *service* generally describes the interface or role and *component* describes the interface + one or more implementations. But that's just me. -farra

Stephen adds:

I disagree with this. A "component" is an implementation artifact that exposes 0..n services. A "service" is computation contract exposed by a component. A component may include many other features that are not exposed through the services that is publishes.

A "service" is typically represented by a Java interface and possibly supporting meta-info (such as a version, attributes, etc.).

A "component" is an example of a "service-delivery-strategy".

Fortress and Merlin arrive

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

Fortress supports legacy ECM components but provides a number of features like basic meta-data configuration (instead of a "roles" file), dynamic service activation, lifecycle extensions, instrumentation support and so on. 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 seperation between container concerns and component concerns. As such, all Merlin applications are never dependent on any actual Merlin code (at least in order to compile). A new meta data model was developed, hierarchical block support added, and Merlin provided support for standalone or embedded environments.

For the sake of completeness, we should also mention **Tweety** which was a very basic container developed for the sole purpose as a teaching tool. Tweety has since been abandoned, but not quite forgotten...

Container Bonanza

Now Avalon was left with a horde of containers: Phoenix, Merlin, Fortress and the deprecated ECM. After a long debate on the merits of one or many containers, the decision was made to push for Avalon unification. The grand container to rule them all would one day become a reality. To reach this goal, Merlin has been selected as the "flagship" Avalon container and that is where you will find most of the action today. Merlin supports a host of features and configurations, but there's still work to be done to fully support features still only found in Fortress or Phoenix.

The Saga Continues

In 2003 and early 2004, there is a plenty of infighting within the Avalon community, resulting in one of the core developers being expelled, Phoenix to fork into Loom (at loom.codehaus.org) and finally a vote for "A Single Unified Platform", which passed. The vote put the lid on for a while, but not for long. More infighting arose, and various solutions were presented, which eventually resulted in the creation of Excalibur top level project (excalibur.apache.org). Excalibur TLP took over the development of Excalibur and Fortress (Phoenix had evolved too much as Loom to be bothered to 'come back'). This allowed Avalon to concentrate on its mission to provide solutions for truly inter-operable components.

One Specification, One Reference implementation

So the choice has become simple. Avalon will continue to provide solutions to make life easier for the application developer, by truly reusable and inter-changable components. The specifications still need a lot of work, but the reference implementation, Merlin, is a stable and capable container. For users the question "Which container?" no longer needs to be asked. We can all pull together and work on the Single Unified Platform, without redoing the work for several containers, each with their own variations and quirks.