AvalonContainerFAQ

General Avalon Container FAQ

- What is the responsibility of the container?
  - As a minimum - a container is responsible for the establishment of a set of services. This responsibility generally involves making sure that components that provide the services are properly deployed (i.e. instantiated correctly with respect to the component lifestyle, and deployed in accordance with the component's lifecycle criteria and related deployment directives).
  - Most containers go further than this minimum by providing different means to access services or by providing management access.
- What is the relationship to the service manager?
  - A ServiceManager is simply one of several artifacts that are used by a container to provide content to a component implementation.
- Is it the container's job mainly to configure the service manager? Or does the container serve different purposes depending on the framework implementation?
  - A client (component implementation) should not be concerned with how and what a container does to establish service provided by a service manager. In particular, different containers approach the service delivery problem in different ways. Some containers resolve components on demand based on the arguments supplied to the lookup argument, and others wire together a solution in advance.
- If the service manager controls the lifecycle of its services, what dictates when those events are called? I didn't see any lifecycle events on the service manager itself, so when are stop(), suspend(), ect being called?
  - A fully deployed component will normally be taken through its deployment lifecycle by a container before a reference is supplied to the client.
  - I.e. the component providing the service will have already been started before the client component receives the reference to the service.
  - Things like suspend() are a lot more complicated and support across containers for this is fragmented at best (basically additional work is needed to handle suspension and resumption in a predictable manner).
- Also, is the service manager the main access point into the avalon framework by the rest of the application?
  - It is one of several - but probably the most important. The others include Context, Configuration, Parameters, etc.
- Are components and services treated differently internally by the service manager? If not, why the distinction?
  - A component is an implementation artifact whereas the service is generally an interface implemented by the component.
  - Note - this is the general case, however, containers exist that enable the establishment of services using means other than strait implementation of an interface.
- I've looked at excalibur, phoenix, and merlin, and it gets a little confusing about the relationships and responsibilities of components, services, service managers, component managers, containers, container managers, etc.
  - component == implementation artifact
  - services == generally speaking - the interface a component exports
  - service manager == a utility used by a container when communicating with a component
  - containers == different things to different people but generally takes care of lifestyles and lifecycles of components
- The service manager would know how to gracefully shutdown every service it manages. Is that close to being accurate?
  - The container (behind a service manager) receives information about the release of services. A container may choose to dispose of the component following a release depending of the lifestyle of the component providing the service and the status of other dependencies that the container has established for the component. Termination of the container should result in the orderly decommissioning of all established components. Different containers handle this function in different ways.
- What is Phoenix, Merlin and Fortress, are they essentially the same, are they all containers?
  - yes, see http://avalon.apache.org/product/containers/
- If so which one should I choose?
  - As always, it depends on what you want to do. Each container has its niche. Phoenix and Fortress have stable releases on the Avalon download site, Merlin release candidate (RC) binaries are also available.
  - Phoenix's strong point are server-like applications. That is, if you want to build a web server, and FTP server, a new EJB container, a chat server, and so on, you may want to try Phoenix. It's stable and there are plenty of existing Phoenix blocks (applications) you can use as starting points. That said, you can also use Phoenix in standalone swing applications, but that's often easier to do with Fortress.
  - Fortress is a light-weight fully-compliant Avalon 4 container. It's often called 'embeddable' partly because it doesn't come with its own bootstrapper. That is, you'll need to write your own 'main' class to start the container. The advantage is that you can easily start it up in just about any environment – servlets, swing apps, command line apps, etc. It doesn't have all the bells and whistles of Phoenix or Merlin, but it's solid and easy to use. Fortress also replaces the older Excalibur Component Manager, so if you have legacy code using ECM, check out Fortress.
  - Merlin is the latest and greatest and where a lot of the current action is. Currently (late 2003), you'll have to settle for beta releases only, but it is certainly usable and has a number of advanced features that you might find hard to live without. Certain Avalon developers will tell you that you shouldn't bother with anything else, but there's plenty of documentation and several small tutorials available for you to make your own decision.
- If I wrote some kind of network daemon, using TCP/IP connections I could use lots of stuff in excalibur. How can I establish a kind of architecture where some of the classes can be reloaded into the system without stopping the daemon?
  - This requires a classloader above and beyond the resources available in the JDK or any of the Avalon containers.
  - Basically, the easiest way to do what you describe is to you use JMX or a JMX-based interface to tell phoenix to do some kind of modification of your server application. Rather good docs on this @ http://avalon.apache.org/phoenix/mx/index.html
- which ones (components) can really be used in Fortress, if they were originally written for ECM or Phoenix
  - If they were written for ECM, all of them. If they were written for Phoenix, it somewhat depends on how it depends on Phoenix. Most of the time it is due to the "BlockInfo" or "Block Context" dependencies--which can easily be worked around.
- which ones are stable - the current package shuffeling and the lack of an documented overview makes it difficult to explain
  - True. We are going through and making releases of all the stable stuff. Basically if it is accessible from http://avalon.apache.org/bindownload.cgi then it is stable.
- Does the one of the containers has any support of transaction management as an EJB Sever offers with JTA/JTS? Is there any support for concurrency available?

Container Overview

Phoenix

- basically oriented towards an app-server
- manages deployment of a set of "applications"
- "applications" are composed of components that phoenix assembles based on meta-info and meta-data
- is released, has some good features like JMX
- is limited to non-embedded applications
- provide support for component packaging.
- http://avalon.apache.org/phoenix/
- pre-releases binaries (phoenix and apps (like demo)) http://cvs.apache.org/~hammant/
- status (2003/03/28) : 4.0.3 stable, new version pending

Merlin

- next generation container - leverages and enhances the notion of meta based management from phoenix
- provides support for dynamic component assembly and deployment
- manages a set of "blocks" (where a block is a container heirachy that can contain components and other blocks and is itself a component)
- containers represent the runtime management environment for the set of components contained within then and handle the orderly startup and shutdown
- full support for different component lifestyles
- suppport for pluggable lifecycle strategies
- fully embeddable
- provides a URL based scheme for service resolution
- provide support for component packaging. Examples of Merlin based packaging is included in a bunch demos that are included in the CVS (just do a checkout of Avalon, Excalibur and Sandbox, build merlin, then $demo).
- Merlin lets you do dynamic addition to the classloader but I (Stephen McConnell) haven't tried "reloading" a classes that has changed - my gues is that some work would be needed on the type, profile, and appliance libraries (but probably not too much).
- http://avalon.apache.org/merlin/
- http://nagoya.apache.org/wiki/apachewiki.cgi?Merlin
- status (2003/03/28) : unstable

Fortress

- a container that handles dynamic service activation (as opposed to predicative in the case of Phoenix and Merlin)
- container model handles components
- oriented towards the pooled component environment but provides a full set of compnonent lifestyle policies
- fully embeddable
- uses a service locator model
- http://avalon.apache.org/excalibur/fortress/
- alpha binaries :
    - http://avalon.apache.org/~bloritsch/excalibur-dist/
    - http://cvs.sourceforge.net/cgi-bin/viewcvs.cgi/d-haven/guiapp/lib/
    - http://cvs.sourceforge.net/cgi-bin/viewcvs.cgi/d-haven/guiapp/tools-lib/ (tools)
- status (2003/30/08) : released

Tweety

- alpha binaries : http://cvs.apache.org/~leosimons/excalibur/tweety/
- status (2003/03/28) : education purpose only

Excalibur Component Manager (ECM)

- http://avalon.apache.org/excalibur/component/
- status (2003/03/28) : stable, but deprecated

Plexus

- http://plexus.codehaus.org/
- http://cvs.codehaus.org/viewcvs.cgi/?cvsroot=plexus

Roadmap

As of November 2003, most development is going into Merlin 3.0 which is targeted as the new 'flagship' Avalon container. However, Phoenix and Fortress are still available, supported and perhaps most importantly, stable.

Return to ["AvalonFAQ"]

(this page is part of the wiki materials for ApacheAvalon; Avalon main page in the wiki is AvalonProjectPages)