AvalonFiveInterfaceProposal

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

The AvalonFive Interface Proposal

When we are separating code needed for compilation vs. code needed for running a system, it makes sense to separate the implementation from the interface definition. For this reason, I would like to set up Avalon5 so that the person developing for it will have a jar with a set of interfaces only. Not one implementation at all.

Why, you ask? The actual implementation of the lifecycle interfaces are many times container specific. In fact, some interfaces in Avalon 4 have no real meaning outside of a container such as ServiceManager or Context. They represent what the container makes available to a component. Therefore the implementation is highly container dependent. The fact that most of our containers do not use the "default" implementation in the JAR leads to support that oppinion. It also allows us to have focused "support" JARs.

Focusing Support

The Logger interface allows us to successfully abstract the Logger from the components that we write. The fact that one component was written against a container using LogKit doesn't mean that it will be deployed that way. A side affect of the current Avalon 4 structure is that we currently need Log4J, LogKit, and JDK 1.4 Logging in the classpath to compile for a distribution. It would be better if the backing logger abstraction was added in a separate JAR complete with a LogManager that the Container uses to find the exact Logger implementation necessary.

We might consider leaving the "braindead" versions of the loggers for debug purposes (i.e. NullLogger and ConsoleLogger). They can be placed in a subpackage called debug or something.

It will also allow us to add support for certain environments. For example, we might want a specific contract for containers embedded in a Servlet context--i.e. where the values come from for standard entries and accessing configuration values from the ServletContext, etc. By providing a "ServletSupport" JAR, we can have the proper Logging formatters to glue into a Servlet environment as well as provide proper context support for that environment.

Pluggable Contracts

The concept of "pluggable contracts" is the process of having a well defined *set* of contracts that we can use in a given environment. We always need the *constant* or *guaranteed* contracts to act as a base. The contract sets that get superimposed on the base contracts are constant for the environment. If you change the environment, you get a new set of contracts.

At first this might seem contradictory to providing support for cross-container components. However, when applied correctly, it helps containers provide a consistent environment to the component regardless of the environment.

Let's look at Cocoon for example. In a servlet environment, it has added some abstractions to keep the components isolated from the javax.servlet API so that it has the potential of working from the command line and the Servlet environment. The constant contracts are those guaranteed by the interface abstractions. The pluggable ones are those that map those constant contracts to the environment. The command line environment provides a consistent set of contracts so that the user knows how to customize it for their use. The same goes for the servlet environment. The pluggable contract sets are for how the container maps the environment to the component, not for additional functionality supplied to the component from the container.

By standardizing component contracts strictly from the viewpoint of the component (i.e. interfaces) we provide a standard in which the component author can feel confident that their component can work anywhere.

By standardizing component contracts strictly from the viewpoint of the environment to the container, we provide a standard mechanism that any Avalon user will be able to use to customize the environment. No extra coding is necessary, the container can simply use the JAR that provides support to the environment and everything is happy. those defined by the interfaces.