

JavaBeansVsComponents

<http://jakarta.apache.org/avalon/images/header.gif>

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

JavaBeans Vs. Avalon Components

[JavaBeans](#) were Java's first foray into component based design. They trade the traditional separate interface for a set of design idioms. Idioms help to define some rules and regulations for creating classes that are easy to tool, but they lack the ease of use of traditional components.

Avalon Components use the standard component design idiom of using a separate interface from the implementation. In a pure OOD arena, this is called the "Bridge" pattern. In Component Based Design (CBD), it is merely how you define a component.

Avalon Components

Avalon Components follow two design principles: Inversion of Control (IOC) and Separation of Concerns (SOC). The approach allows a component writer to only use the information that they want to use. It also makes the components listeners for events that come from the container. Those events tell the component that it is going through a particular stage in its lifecycle.

Avalon Components are very easy to work with because it has a defined lifecycle. You know when the component is in use, and when it is being created or destroyed. Because we know when important events happen in a component's life, we can minimize the amount of thread contention that a component has to manage. It also encourages components that can be run using several threads.

JavaBeans

[JavaBeans](#) were originally introduced as graphical components. They really shine in this role, and work very well. The design idioms that make up [JavaBeans](#) have been applied to server side technologies like `javax.sql.DataSource` objects developed by database vendors. [JavaBeans](#) on the server side present a number of design problems:

- Concurrency problems due to changing configurations at runtime
- Impossible to tell the [JavaBean](#) when configuration is finished
- Requires special event processing to serialize access or changes to the [JavaBean](#)
- Use of introspection during configuration is slow (see [JavaBeansVsConfiguration](#)).

Advantages of Components over [JavaBeans](#)

Avalon components are designed for silent operation behind the scenes. In fact, they work best for the "Controller" portion of the Model-View-Controller design pattern. [JavaBeans](#) are designed for vocal interaction with the user. They tend to favor either the "Model" portion of MVC, or the "View" portion.

- It is easier to develop server side components using Avalon than the older [JavaBean](#) standard.
- Avalon components are truly components. [JavaBeans](#) do not separate usage contracts from the implementation code.
- Avalon components are more resilient to change due to the fact that component contracts are well understood, and are separate from the implementation code.
- Concurrency management for Avalon components is easier than for [JavaBeans](#) because there is a specific time when the component can be configured--contrasted with [JavaBeans](#) which has no control.

Advantages of [JavaBeans](#) over Components

Due to all the workarounds that [JavaBeans](#) have for their shortcomings, they have better tooling. [JavaBean](#) tooling comes shipped with the JDK used to run the beans. The tooling is in the `java.beans` package, as well as the reflection package. The Swing API is full of [JavaBeans](#) you can use in your own software. Some of the more mature aspects of [JavaBeans](#) has to do with the event structure associated with it.

- It is easier to develop client side building blocks (they are not true components) that interrelate to each other.
- The "Publish-Subscribe" design pattern used for event notification is advanced--and allows new software to participate in the interaction without having to change the original code.
- Property change notification is part of the more generic event notification, but it allows all parties monitoring the contents of a [JavaBean](#) to react when it is altered.

Summary

[JavaBeans](#) are useful for modeling information in a very strongly typed manner. They are also useful for rendering that same information to the user of the application and allowing the user to interact with the rest of the program. They fall short when you attempt to use them to manage the business logic of an application--especially one that is designed to work with several clients simultaneously (client/server).

The event notification system built into the [JavaBeans](#) infrastructure allows you to design a very rich user experience. Avalon recently got outfitted with a more powerful command processor and event routing system. The question remains whether we want to integrate that into Avalon proper. That would allow us to have a SEDA like infrastructure for applications that demand highly scalable and massively concurrent communications.

We need to compare the more powerful features of [JavaBeans](#) to the more powerful features in Avalon to make a fair judgement.