Component Architecture

Table of Contents

[Client-Side and Server-Side Components 3](#_Toc100617940)

[Existing Approaches 3](#_Toc100617941)

[Benefits 4](#_Toc100617942)

The **component architecture** centers around decomposing the overall design into a set of functional components or **modules** that interact with each other to achieve the end goal.

Each component encapsulates the functionality and behavior of a software element into a **reusable** and self-deployable binary unit. These components can act on their own, or interact with other components.

All components live inside a **container**. This container provides the components their context, creates them, manages them and glues them together. It provides lifecycle management, security, deployment, the runtime as well as services for the components.

We have already seen one container (the Web Container), and soon, we will see another (the EJB Container).

A well-defined component architecture is a set of **standards**. The standards must contain specifications for both the container services as well as APIs (classes, interfaces, methods, etc.).

We use development tools like IntelliJ to develop, configure and package components. The components reside on the application server, which provides the runtime environment. Such servers include Tomcat, WildFly, etc.

## Client-Side and Server-Side Components

We can have components on both the **client-side** and the **server-side**.

Client-side ones provide GUIs where users can provide inputs which get directed to the server. There may also be some pre- and post-processing here. Packages like Java Swing, JavaFX, etc. provide client-side GUI components.

Server-side components provide services, such as providing dynamic content, accessing databases, authentication, subscription, etc. These components are provided by Java Servlets, JSPs, EJBs, etc.

## Existing Approaches

There are several existing approaches to the component architecture.

* COM, DCOM, .NET from Microsoft
* CORBA from OMG
* Java EE a.k.a. Jakarta EE from Oracle and Eclipse
* Web Service Technologies like WSDL, XML, UDDI, SOAP and HTTP

## Benefits

* **Ease of Deployment** – New components can just replace older ones without affecting the rest of the system. This can be done since the components use well-defined interfaces.
* **Reusable** – The same components can be used in several applications.
* **Reliable** – Each component can be thoroughly tested easily, which makes them more reliable. This in turn makes the system more reliable.
* **Independence** – Each component being independent makes it easier to develop multiple components in parallel using different developers.