Component

Definitions

A component is a nontrivial, nearly **independent**, and **replaceable part** of a system that fulfils a clear function in the context of a well-defined architecture.

A component conforms to and provides the physical realization of a set of **interfaces**.

A runtime software component is a **dynamically** bindable package of one or more programs managed as a **unit** and accessed through documented **interfaces** that can be discovered at runtime.

A software component is a unit of composition with **contractually specified interfaces** and explicit context dependencies only. A software component can be **deployed** independently and is subject to third-party **composition**.

A business component represents the software implementation of an autonomous business concept or business process. It consists of the software artefacts necessary to express, implement, and deploy the concept as a **reusable element** of a larger business system.

A component is a unit of distributed program structure that **encapsulates its implementation** behind a strict interface comprised of services provided by the component to other components in the system and services required by the component and implemented elsewhere. The explicit declaration of a component's requirements increases reuse by decoupling components from their operating environment.

UML (preferred definition and characteristics), 11.6.3.1 Components

A Component represents a modular part of a system that **encapsulates its contents** and whose manifestation is **replaceable** within its environment.

A Component is a **self-contained unit** that encapsulates the state and behaviour of a number of Classifiers. A Component specifies a **formal contract** of the services that it provides to its clients and those that it requires from other Components or services in the system in terms of its **provided and required Interfaces**.

A Component is a substitutable unit that can be **replaced** at design time or run-time by a Component that offers **equivalent functionality** based on compatibility of its Interfaces.

As long as the environment is fully compatible with the **provided and required Interfaces** of a Component, it will be able to **interact** with this environment. Similarly, a system can be extended by adding new Component types that add new functionality.

A Component is modelled throughout the development **life cycle** and successively refined into deployment and run-time.