Distribution Of Components

[ComponentDesignPatterns | CategoryPattern]

Context

Some of the most difficult challenges that architects and builders face in the creation of enterprise- or Internet-scale software is how to determine where components should be located, and how to deal with change. Changing requirements and technology climate make adapting and evolving software a complicated task. This pattern provides helpful themes in the distribution of components during system construction and operation.

Problem

While the component-based approach to software construction provides many benefits and can facilitate rapid delivery, complexities in architecture, design, development, and operation make it difficult for builders and assemblers to compose software that always meet requirements.

Forces

- Monolithic programs make code reuse difficult and make it hard for teams to coordinate change.
- Designing a new system or breaking an existing up into too many components can result in overly complex and defective systems. BigBallOfMud often results.
- Too much application logic in the client application (also known as TooMuchGuiCode) increases complexity, hardening arteries and causing high blood pressure in fat laboratory mice.
- But totally thin clients that are completely dependent upon servers make these servers get really large and cause them to manage state for every client. This results in distribution, scalability, and reuse problems.

Solution

Distribute the usage of components across all tiers of the application and:

- Build clients that reuse lots of smaller components that share connection resources to access application services. Instances are typically created and destroyed often. They are typically graphical in nature and do not communicate directly with other components; instead, glue is used to tie them together.
- Build servers that consist of a defined set of larger, reusable components plugged into an application infrastructure that supports fault tolerance, is scalable, and pools server-side connection resources with clients. Instances are typically shared across many client sessions. They are typically not graphical and communicate with other services and external legacy and data services.
- Wrap legacy systems, data services, and other external systems or data feeds with components. Connection with these components can be shared to conserve resources.

- Distribute your components in terms of an architecture reference model such as the ThreeTierDistributionArchitecture, the FourLayerArchitecture, or the peer-to-peer distribution architecture.

Resulting Context

An application consisting of a distribution of components can use a LayeredComponentFramework so that components can be built in terms of frameworks that raise programming to a higher level of abstraction. This can be performed in any tier or layer of the component-based system. A BypassableAbstraction can allow programmers more freedom where custom development is required, and LocationTransparency can be used when scalability and flexibility are required. Disregard for component location can lead to assumptions in performance or reliability that can decrease the value of TransparentDistribution. ProcessBoundary can be used in to closely analyze location dependencies and make decisions about the physical location of components early on in the project lifecycle.

Today, we see lots of applications with small visual components plugged into client-side user interfaces that are built in web pages, Visual Basic, Visual C++, Visual Cafe, Visual Age for Java, and many other development tools that facilitate ComponentBasedDevelopment.

Increasingly, we're seeing more emphasis on server-side components as well. They're not as fine-grained, usually bigger, and have a different kind of lifecycle and relation to their infrastructure and other components. With the maturing of technologies like Java Servlets, EnterpriseJavaBeans, CORBA OTM, and MicrosoftDotNet (or will it be MicrosoftIndigo ?), it will become easier to plug in reusable server-side components.

Check out ServiceOrientedArchitecture as an example of the latest (circa 2003) evolution in concepts related to deployment of technologies, and thereby lay out the components necessary for the delivery of services.

DistributionOfComponents, coupled with appropriate decisions regarding location and transparency of components, is an important basis for the rest of ComponentDesignPatterns.

CategoryComponents

Last edit January 4, 2012