DSS Introduction

Decentralized Software Services (DSS) is a lightweight .NET-based runtime environment that sits on top of the Concurrency and Coordination Runtime (CCR). Decentralized Software Services (DSS) provides a lightweight, state-oriented service model that combines the notion of representational state transfer (REST) with a system-level approach for building high-performance, scalable applications. In DSS services are exposed as resources which are accessible both programmatically and for UI manipulation. By integrating service isolation, structured state manipulation, event notification, and formal service composition, DSS addresses the need for writing high-performance, observable, loosely coupled applications running on a single node or across the network.

A primary design goal of DSS is to couple performance with simplicity and robustness. This makes DSS particularly suited for creating applications as compositions of services regardless of whether these services are running within the same node or across the network. The result is a flexible yet simple platform for writing a broad set of applications. DSS uses Decentralized Software Services Protocol (DSSP) and HTTP as the foundation for interacting with services. DSSP is a lightweight SOAP-based protocol that provides a clean, symmetric state transfer application model with support for state manipulation and an event model driven by state changes.

The DSS runtime provides a hosting environment with built-in support for service composition, publish/subscribe, lifetime-management, security, monitoring, logging, and much more both within a single node and across the network. Services can be written in Visual Studio, or using Microsoft Visual Programming Language (VPL). VPL can be used to create applications as compositions of services simply by dragging and dropping them onto a canvas and wiring them together based on their data-dependencies. In addition, DSS Manifest Editor provides a graphical environment for wiring-up, configuring, deploying, and running DSS applications on a single node or across the network.

Problem Areas Covered by DSS

In the following we describe three common problem areas of application design and how DSS addresses them:

Robustness

In any complex system, a failure in a sub-part of the system has the potential of bringing down the whole system. The reason is that a partial failure can lead to a catastrophic failure if it cannot be properly isolated, detected, and handled. Loose coupling is a design pattern that often is invoked as a way of limiting the impact of partial failures. However, in order to build loosely coupled systems, each component must be isolated from all other components as well as from the underlying runtime environment. Two common ways

in which systems fail to isolate components are data isolation and execution isolation. Lack of data isolation can cause the internal state of a service to be corrupt and lack of execution isolation can cause a component to become unresponsive. DSS isolates services in both data and execution. Data isolation is achieved by fully cloning all messages exchanged between services (including system services). The cloning code is generated as part of the compilation and is much faster than full serialization.

Composability

The requirement of robustness forces applications to become compositions of loosely coupled components. This raises the additional problem of how to identify, locate, and compose such components into a running application. Most traditional systems define an application as a single process and not as a composition of loosely coupled services working together leaving the task of composition to the application designer. DSS provides both protocol and runtime support for creating, managing, deploying, and running applications that are composed of loosely coupled services: A service is created and wired-up at runtime based on a description of which other services it needs to compose with in order to function properly. This model allows applications at runtime to configure where each service instance is running. Further, the relationships that each service instance has with other service instances is exposed through DSSP making it possible to see which other service instances any particular service instance is dependent on.

Observability

A critical aspect of any application is that it is possible to know what it is doing, what state it is in, and how it got to be in that state. In short, without a way to observe a system it is impossible to know whether the system is functioning properly. DSS puts the notion of observability at the core of its application model by defining a service as a resource identified by a URI and with exposed state that change as a result of internal service behavior and through interaction with other services. However, the notion of observability goes beyond just monitoring an application by inspecting the state of its services. By uniformly exposing all services in terms of their state, DSS provides simple and consistent solutions for dealing with administration, management, security, and service composition based on state manipulation. Further, DSS enables every DSS service to be accessed through a Web browser enabling rich UI as well as easy integration with a variety of common tools and platforms.

Getting Started

As part of DSS we provide a wide variety of samples and tutorials that illustrate how to get started writing applications ranging from simple "Hello World" to writing complex applications that seamlessly can run on the same node or across the network.

© 2012 Microsoft Corporation. All Rights Reserved.