Distributed Protocol Factory 1.0 Requirements Specification

1. Scope

1.1 Overview

This component provides a factory for accessing a distributed protocol and a default protocol for supporting distributed systems. This protocol is a data transfer protocol for ensuring all nodes in a group receive an update, or none receive the update. It is intended to be used by a variety of application types. The component will support a factory method for loading the protocol, so that specialized protocols can be substituted for the default protocol, when necessitated by the needs of the application.

1.2 Logic Requirements

1.2.1 Protocol Factory

A factory method will be used to allow the application to select a data synchronization protocol at run-time.

1.2.2 Distributed Data API

An API will be defined in an interface that exposes the functionality for an application to make use of this component for distributed data management.

1.2.3 Message Broadcast

The API will support a means for a node in the group to broadcast changes to other nodes. The calling application will be notified if the broadcast was successfully received or not.

1.2.4 Message Receipt

The API will support a means for nodes in the group to receive and confirm messages.

1.2.5 Error Notification

The API will support a means for an application to notify a message sender that a message received was acted on and caused an error to occur on a remote server.

1.2.6 Protocol Decoupling

The API will not require the application to have knowledge of implementation details. Including:

- Identities of other servers
- Entry/Exit of other servers from the group
- Underlying protocol layers (e.g. TCP/IP or UDP, etc.)
-

1.2.7 Application Re-sync

The API will allow a node to re-sync with the group when it comes back online.

1.2.8 Group Division

The API will provide a means for an application to switch groups. NB: This should be done in a manner consistent with requirement 1.2.5

1.2.9 Group Move

The API will support moving a server from one group to another and having that server re-sync with the new group.

1.2.10 Default Implementation

A default implementation of the protocol will be provided with the component.

1.2.11 Fault Tolerance

The default protocol implementation must support failure of one or more nodes in the group (simultaneously or otherwise).

1.2.12 Scalability

The protocol will provide functional scalability across multiple servers; i.e. the algorithm logic is as correct for n+1 servers as it is for n. (Expected performance bottleneck points should be identified.)

1.3 Required Algorithms

The design of the component should specify algorithms necessary for guaranteeing broadcasts are received by all or none of the nodes.

1.4 Example of the Software Usage

Example 1 – Distributed Cache
The current version of the Distributed Simple Cache implements a protocol that deadlock if more than half of the servers in a group fail. This component would be used instead to decouple the implementation of the synchronization from the cache. Depending on an application's requirements, different protocol implementations could be used that optimize for various attributes, such as speed or stability.

Example 2 – Distributed Configuration Manager
A new persistence layer for the configuration manager could be built that allowed synchronization across multiple application servers. This layer would use the default protocol implementation to guarantee that all application servers were received the same updates to their configurations. If the environment was changed from non-clustered to clustered application servers, then a new protocol could be built that utilizes app server clustering properties to more effectively handle the synchronization.

1.5 Future Component Direction

Protocols that are highly tailored to specific target environments or that use different assumptions, could be built and offered as part of this component.

2. Interface Requirements

2.1.1 Graphical User Interface Requirements

None.

2.1.2 External Interfaces

The design of this component will identify any necessary underlying protocols (such as tcp/ip or udp.)

2.1.3 Environment Requirements

- Development language: C#
- Compile target: .NET 1.1

2.1.4 Namespace Structure

TopCoder.Network.Synchronization

3. Software Requirements

3.1 Administration Requirements

3.1.1 What elements of the component need to be configurable?

The protocol implementation should be configurable programmatically.

3.2 Technical Constraints

3.2.1 Are there particular frameworks or standards that are required?

As determined by design.

3.2.2 TopCoder Software Component Dependencies:

None. (Do not use configuration manager.)

3.2.3 Third Party Component, Library, or Product Dependencies:

None.

3.2.4 QA Environment:

- Windows 2003

3.3 Design Constraints

The component design and development solutions must adhere to the guidelines as outlined in the TopCoder Software Component Guidelines. Modifications to these guidelines for this component should be detailed below.

3.4 Required Documentation

3.4.1 Design Documentation

- Use-Case Diagram
- Class Diagram
- Sequence Diagram
- Component Specification

3.4.2 Help / User Documentation

- Design documents must clearly define intended component usage in the ‘Documentation’ tab of Poseidon.