**Software Design Document**

for

Ragade’s Cube Networking Component

Written By: Chris Boyle

Last Modified: October 14, 2007

Table of Contents

[Introduction 4](#_Toc180204296)

[Purpose 4](#_Toc180204297)

[Scope 4](#_Toc180204298)

[Definitions and Acronyms 4](#_Toc180204299)

[Definitions 4](#_Toc180204300)

[Acronyms 4](#_Toc180204301)

[References 4](#_Toc180204302)

[Decomposition Description 4](#_Toc180204303)

[Module Decomposition 4](#_Toc180204304)

[Common Utility Description 4](#_Toc180204305)

[Server Description 5](#_Toc180204306)

[Client Description 5](#_Toc180204307)

[Concurrent Process Decomposition 5](#_Toc180204308)

[Server 5](#_Toc180204309)

[Client 5](#_Toc180204310)

[Data Decomposition 5](#_Toc180204311)

[Logical Response Message 5](#_Toc180204312)

[Player Request Message 6](#_Toc180204313)

[Game Request Message 6](#_Toc180204314)

[List Game Response Message 6](#_Toc180204315)

[Game Start Request Message 6](#_Toc180204316)

[Game Play Request Message 7](#_Toc180204317)

[Game Play Action Message 7](#_Toc180204318)

[Game Play Move Buffer Message 7](#_Toc180204319)

[Game Play Response Message 7](#_Toc180204320)

[Dependency Description 7](#_Toc180204321)

[Inter-module Dependencies 7](#_Toc180204322)

[Inter-process Dependencies 8](#_Toc180204323)

[Data Dependencies 8](#_Toc180204324)

[Interface Description 8](#_Toc180204325)

[Module Interface 8](#_Toc180204326)

[Common Utility Interface 9](#_Toc180204327)

[Abstract Server Interface 10](#_Toc180204328)

[Server Implementation Interface 11](#_Toc180204329)

[Abstract Client Interface 12](#_Toc180204330)

[Client Implementation Interface 12](#_Toc180204331)

[Process Interface 13](#_Toc180204332)

[Use Case 1: Sending a message to a Client from Server (or Server from Client) 13](#_Toc180204333)

[Use Case 2: A Client should be able to connect/disconnect to a server. 14](#_Toc180204334)

[Specific Implementation 15](#_Toc180204335)

[Common Utility 15](#_Toc180204336)

[Client 15](#_Toc180204337)

[Server 16](#_Toc180204338)

# Introduction

## Purpose

The purpose of this document is to describe the design of the Networking subsystem for the Ragade’s Cube Game project. It will contain a description of the client-server architecture and its corresponding data message formats.

## Scope

This document is to be used by testers and programmers of the networking component. A thorough description of all the functionality is provided for the overall software design. The specific implementation of the functionality is dependent on the programmer of the component.

## Definitions and Acronyms

### Definitions

* Ragade’s Cube – The Rubik’s Cube game that this document is designed for.
* Client/Server - In communications, the model of interaction in distributed data processing in which a program at one site sends a request to a program at another site and awaits a response. The requesting program is called a client; the answering program is called a server.

### Acronyms

* RC – Ragade’s Cube

# References

* Lidgren Network Library - <http://code.google.com/p/lidgren-library-network/>
* MSDN Sockets. <http://msdn2.microsoft.com/en-us/library/system.net.sockets.socket.aspx>

# Decomposition Description

This section will describe the different sub-component of the networking component.

## Module Decomposition

The networking component can be divided into three main modules: the Client, the Server, and the Common Utility. The reason to separate the functionality is because it is unnecessary for the client and server to use any of the specific implementations provided for each.

Each main module contains its own sub-modules that allow for a more customized design based on the specific implementations.

### Common Utility Description

The common utility is compiled as *RC.Networking.dll*. The purpose of the module is to provide a common location for shared functionality between the client and the server. This utility includes the basic abstract implementations for the Server and Client, as well as the definitions for all possible data messages that can be transmitted between the Server and the Client.

### Server Description

The server is represented by two separate compiled entities: *RC.Networking.Server.dll* and *RC.Networking.Server.Console.exe*. The compiled DLL will contain all of the server logic that can facilitate a Ragade’s Cube networked game. The DLL will not contain the specific code that allows connections to be handled because that will be handled in the EXE. Along with containing the connection handlers, the EXE will initialize the server to allow game functionalities to be utilized during the server operation.

### Client Description

The server is represented by two separate compiled entities: *RC.Networking.Client.dll* and *RC.Networking.Client.Impl.dll*. The purpose of the two DLLs is to allow generic client functionalities to be separate from the specific implementation. The implementation DLL is to be used in the Ragade’s Cube executable in order to allow multi-player games.

## Concurrent Process Decomposition

The networking component will contain two concurrently running processes: the Server and the Client.

### Server

The server is to run on a computer that has a network connection. The purpose of the server is to have a central hub that can manage multiple clients to play the Ragade’s Cube game concurrently. A single instance of the server is to exist and handle up to a default value of 1000 concurrent client connections. The server will receive messages from each client and response to their requests accordingly.

### Client

The client is to run on a Ragade’s Cube user and has the ability to communicate with the server. The game logic will provide various messages that need to be communicated to the server.

## Data Decomposition

There are various messages that have to be handled with both the Client and the Server. The messages defined below describe their form. Each message is implemented into a corresponding DataMessage abstract class. The data message class is used to encapsulate the data that is transmitted to the Clients and the Servers.



Data message class.

### Logical Response Message

The logical player response message informs the Client user that a player’s request operation was successful.

* Operation Success. Provides a Boolean value that determines if the last operation requested by the user was successful.
* User Name. This is the username that is used to identify the user.

### Player Request Message

The player request message informs the Server that a new player has been created on a Client. The player must be given a unique name that the Server will check and respond with a logical response message.

* Request Type. A request type defines the type of request by the client. The types of requests that the client can have are defined here:
  + Register. This registers the user on the server. A logical player response message is required.
  + Unregister. This un-registers the user on the server.
* User Name. This is the username that is used to identify the user.

### Game Request Message

A game request message is a message that is sent from the Client to the Server. The Server is then expected to parse the request and send back a corresponding game response message.

* Request Type. A request type defines the type of request by the client. The types of requests that the client can have are defined here:
  + Create. This creates a new game session. A logical response message is required.
  + Join. This joins a user to a game session. A logical response message is required.
  + List. This requests that a user receives a list of possible game sessions to join. The server must respond by providing the server list. A list game response message is required.
  + Quit. This requests that a user quits a current game session. A logical response message is required.
* Game Session Name. This is a possible parameter to define the game session name. This is only important for Create, Join, and Quit request types.
* User Name. This is the username that is used to identify the user.

### List Game Response Message

The list game response message is used to responds to the client the current game session that are available on the server.

* List. The list of current game sessions.
* User Name. This is the username that is used to identify the user.

### Game Start Request Message

This message informs the server that the user is waiting for the game to start. The Client is waiting for game play response messages. A logical response message is required.

* User Name. This is the username that is used to identify the user.

### Game Play Request Message

The game play request message informs the Server about events that occur during a user’s game. A logical response message is required.

* Request Type. A request type defines the type of request by the client. The types of requests that the client can have are defined here:
  + Pause. This pauses the current game session. A game play response message is required.
* Game Session Name. This is a possible parameter to define the game session name.
* User Name. This is the username that is used to identify the user.

### Game Play Action Message

The game play action message defines an action that can occur in a game.

* Move. The move that was performed by the user.
* Game Session Name. This is a possible parameter to define the game session name.
* User Name. This is the username that is used to identify the user.

### Game Play Move Buffer Message

When requested by the Server, the Client must provide a buffer of all moves performed during the game session. This may be required during possible data message errors where another Client may need to refresh a user’s game cube.

* Move Buffer. The move buffer that was performed by the user.
* Game Session Name. This is a possible parameter to define the game session name.
* User Name. This is the username that is used to identify the user.

### Game Play Response Message

This message is to inform the users that a game event has occurred.

* Request Type. A request type defines the type of request by the client. The types of requests that the client can have are defined here:
  + Pause. This pauses the current game session.
  + Finished. This tells the Client that a game session is finished.
* User Name. The user that caused the operation.

# Dependency Description

The successful operation of the networking component is dependent that a valid Server and Client is running concurrently.

## Inter-module Dependencies

Overall, there are five actual modules that contain dependencies within the networking component.



The dependencies.

#### Abstract Modules

* Common Utility (RC.Networking.dll) – No dependencies.
* Abstract Server (RC.Networking.Server.dll) – Common Utility
* Abstract Client(RC.Networking.Client.dll) – Common Utility

#### Implementation Modules

* Client Implementation (RC.Networking.Client.Impl.dll) – Common Utility, Abstract Server, Lidgren Networking Library
* Server Implementation (RC.Networking.Server.Console.exe) – Common Utility, Abstract Server, Lidgren Networking Library

## Inter-process Dependencies

The processes that depend on each other are the Client and Server implementations. The Server has the ability the run independently, but the Client is completely dependent on the Server.

## Data Dependencies

A valid IP address and port must be known for the Client to successfully connection to the Server.

# Interface Description

## Module Interface

### Common Utility Interface

#### NetworkDevice

The NetworkDevice defines the functionality that is provided by both the Client and the Server modules. Every NetworkDevice will contain three important components: message manager, engine, and service container.

The network device has the ability to run it underlying engine to run on a separate thread. The separate thread is used because the engine, whether it is the client or the server, will need to run independently to handle messages and events. Because the user will interact with the NetworkDevice on a separate thread, the system should become more responsive because a request will not affect the user’s thread.

The message manager will be used to define the operations that will be performed when a message is received. The message manager will be defined further below.

The service container will contain various services that are provided by the NetworkDevice. When developing on a particular device, the service container will provide functionality in a more elegant method. This allows certain services to become hidden from a section of code if access should not be provided.

#### INetworkDeviceEngine

The INetworkDeviceEngine defines the interface that is provided for the NetworkDevice to successfully run. The engine should handle all connections, messages, or any other Client/Server specific event.

#### DataMessage

The DataMessage defines an abstract class that represents a message that is sent from Clients and Servers. In its abstract form, the only data member that it contains is the DataMessageHeader.

#### DataMessageHeader

The DataMessageHeader contains information that all data messages must contain. The only information that it contains currently is the timestamp of the message.

#### IMessageRegistrar

The IMessageRegistrar interface defines the interfaces for a class that registers and unregisters event handlers for particular DataMessages that are handled by a INetworkDeviceEngine.

#### IMessageHandler

The IMessageHandler interface defines the interface for a class that handles DataMessages after they are obtained from the INetworkDeviceEngine.

#### MessageManager

The MessageManager class implements both the IMessageRegistrar and IMessageHandler interfaces. It is exposed to other modules by registering itself with the NetworkDevice’s service container. A mapping of delegates to events is obtained and when events occur, the MessageManager makes sure that there is an appropriate response.

### Abstract Server Interface

#### ServerDevice

The ServerDevice implements the NetworkDevice abstract class and defines the operations that are required for a Server. The three basic components that the ServerDevices must contain are the connection manager, player manager, and the game session manager. Each of the managers must exist independently but each update one another in order to keep the data current. Each manager is registered with the device’s service container in order to allow sharing of functionality.

#### ServerEngine

The ServerEngine abstract class implements the INetworkDeviceEngine interface to provide a more specific interface for Server functionality. Still, the ServerEngine is abstract to allow a pluggable engine provider for the ServerEngine. It provides events when connections or disconnections occur in the server.

#### ServerEngineEventArgs

The ServerEngineEventArgs are used with the ServerEngine events. This passes important information about a client that connected or disconnected during the server operation.

#### IServerEngineListener

This defines an interface for a server engine event listener. The interface will define a class that has the ability to listen to all possible events provided by the ServerEngine. This interface facilitates the easy event listener registration process.

#### GameSession

The GameSession class is used to manage created games. The GameSession is responsible for keeping track of the current users in a game. Also, it is given the responsibility to maintain all possible game metrics like running time and the number of moves for each user. The game session should also determine if a game is finished or if a game is currently paused to not allow moves from other users.

#### GameSessionHandler

The GameSessionHandler is responsible for registering all of the messages that are needed by the GameSessionManager in order to successfully manage the game sessions. It registers the messages with the IMessageRegistrar that is provided by the Servers’ service container. It then uses the current GameSessionManager’s interface to perform the correct operation when a DataMessage is present.

#### GameSessionManager

The game session manager receives DataMessages, parses each and transmits the data to the appropriate GameSession. The GameSessionManager is also responsible for the creation and deletion of GameSessions when it is deemed appropriate. The GameSessionManager listens to player deletion events from the PlayerManager to determine if a player has exited current game session.

#### IGameSessionManager

The IGameSessionManager is an interface for the GameSessionManager that makes it available through the Servers’ service container.

#### IConnectionManager

The IConnectionManager is an interface for the ConnectionManager that makes it available through the Servers’ service container.

#### IConnectionChecker

The IConnectionChecker is an interface that defines a class that provides functionality that can check to see if an IP Address that is requesting connection is valid.

#### ConnectionManager

The connection manager is used to register currently connection clients and maintain a record of connected users. The record of connections is saved and used to determine if a particular IP Address is abusing the server and should be denied entry. This class implements both the IConnectionManager and the IConnectionChecker interfaces to make it available through the Servers’ service container.

The ConnectionManager provides events for new connections and disconnections to allow other classes to know when they occur.

#### IPlayerManager

The IConnectionManager is an interface for the ConnectionManager that makes it available through the Servers’ service container.

#### PlayerManager

The player manager is used to maintain a list of currently connected user’s and their connections. The PlayerManager listens to the ConnectionManager’s disconnection event to determine if a player has disconnected from the Server. When players are added or removed from the PlayerManager, they also provide an event to specify that the action has occurred.

#### PlayerHandler

The PlayerHandler is responsible for registering all of the messages that are needed by the PlayerManager in order to successfully manage the game sessions. It registers the messages with the IMessageRegistrar that is provided by the Servers’ service container. It then uses the current PlayerManager’s interface to perform the correct operation when a DataMessage is present.

### Server Implementation Interface

#### Program

This is the class that contains the Main function.

#### ServerImpl

The ServerImpl creates an actual implementation of the SeverDevice abstract class. This is achieved by using the ServerEngineImpl.

#### ServerEngineImpl

The ServerEngineImpl class provides all of the internal functionality for the Server. This is the class that runs on a separate thread in order to handle server events and send messages to clients. This class makes use of the Lidgren Server Library to provide a reliable UDP class library for handling the server events.

When clients are attempting to connect, the IConnectionCheck is used to determine if the connection should be allowed. If the client is finally connectioned, the IConnectionManager is used to register the client. An equivalent operation will occur when a client disconnects.

In the event that a message is received from a client, the IMessageManager is used to handle the messages.

### Abstract Client Interface

#### ClientDevice

The ClientDevice implements the NetworkDevice abstract class and defines the operations that are required for a Client. The only functions that are required for the client are the ability to connect, disconnect, and send messages to a server. Individual message handlers are to be registered through the MessageManager that is provided by the NetworkDevice.

#### ClientEngine

The ClientEngine abstract class implements the INetworkDeviceEngine interface to provide a more specific interface for Client functionality. Still, the ClientEngine is abstract to allow a pluggable engine provider for the ClientEngine. It provides the ability the connection and disconnect to an individual server.

#### ClientEngineEventArgs

The ClientEngineEventArgs are used with the ClientEngine events. This passes important information about a server that connected or disconnected during the client operation.

#### IClientEngineListener

This defines an interface for a client engine event listener. The interface will define a class that has the ability to listen to all possible events provided by the ClientEngine. This interface facilitates the easy event listener registration process.

### Client Implementation Interface

#### ClientImpl

The ClientImpl creates an actual implementation of the SeverDevice abstract class. This is achieved by using the ClientEngineImpl.

Also, the ClientImpl class implements the IRCGameProvider interface that is used to provide and manage games on the client. It maps the functionality from the actual client implementations to the messages that are sent to and from the server.

#### ClientEngineImpl

The ClientEngineImpl class provides all of the internal functionality for the Client. This is the class that runs on a separate thread in order to handle client events and send messages to the server. This class makes use of the Lidgren Server Library to provide a reliable UDP class library for handling the client events.

In the event that a message is received from a server, the IMessageManager is used to handle the messages.

## Process Interface

The process interface.

### Use Case 1: Sending a message to a Client from Server (or Server from Client)



Use Case 1.

This is when the Server decides that it needs to send a message to the Client. The reverse case is also true when the Server is receiving a message from the Client.

The Server decides that message is necessary to be sent, it is then parsed into a form that can be transmitted to the client. The message is then sent through the Server Engine and received from the Client Engine. The Client Engine uses its local MessageHandler to find the appropriate action for the DataMessage.



Sequence of events for Use Case 1.

### Use Case 2: A Client should be able to connect/disconnect to a server.



Use Case 2.

This occurs when the client decides it needs to connection or disconnect to a server. The connection process starts with the ClientDeviceEngine that sends a message that eventually gets handled by the ServerDeviceEngine. The Server then checks to see if the Client’s IP address is valid and adds the connection to its connection list.

The reverse process occurs during the disconnection. The only difference is that the IP Address is not checked during the process.



Sequence of Events for Use Case 2.

# Specific Implementation

## Common Utility

The Common Utility will be used to contain any cross-cutting concern that should be handled in both the Client and the Server.



Class diagram for Common Utility.

The NetworkDevice is the controlling class for all of the functionality. It facilitates the main functions for a Client or Server and allows those functions to be extended through its service container and its message manager.

If the NetworkDevice is to have its functionality extended, the server container can be accessed publicly through the NetworkDevice interface. Additional message handlers can be added through the IMessageRegistrar or a new cross-cutting server can be added to share with other extensions of the device.

The INetworkDeviceEngine allows any type of engine to drive the Client or Server. This design allows third-party Client/Server libraries to be integrated without changing the basic structure of the networking component.

## Client

The client is to provide a non-specific implementation of the ClientDevice.



Class diagram for Client.

The ClientDevice is used to provide an interface to controlling the Client. The ClientDevice exposes functions like, Connect() and Disconnect() to allow it to communicate with only one server at a time. The ClientDevice has the ability to be running without a connection to the Server. If the Client is not connected to the Server, it will not be able to process any new messages until the connection is secured.



State Diagram for Client.

## Server

The server exists to facilitate the communication of clients for a particular game session. Each ServerDevice contains a connection manager, player manager, and game session manager. The ServerDevice is responsible for transmitting messages to all of the clients and making sure that specific game logic is followed.

The GameSessionManager is responsible for maintaining the game users and the game logic.

The PlayerManager is responsible for tracking a player name to the connection that allow communication to the player.

The CommunicationManager is responsible for allowing users to connect to the server and maintaining a current list of connected clients.



Partial Class Diagram for the Server.

In order for the managers on the ServerDevice to run, the service container is used to extend the device’s functionality. The PlayerManager and GameSessionManager have specific messages that has to be handled during operation, so a handler class for each is created to help with the message registration. The ConnectionManager handles connection/disconnection event directly from the server engine.



Partial Class Diagram for the Server.

The ServerDevice is only to run and accept new connections when the device is specified to be running. When this occurs, it then can: accept new client connections and accept message that originate from the clients.



State Diagram for the Server.