EnvironmentInterface – Library for Communicating Between Different Environments

by Glen Berseth

back cover

Contents

1 Introduction

1.1 Message Modularity

1.1.1 Converting messages

When a message is being sent out of the interface the Class that is the connection to the external environment can have a class assigned to it that will attempt to convert the *Message* that is passed to it. To create a different Converting class extend from the *Convert* class and override the convert() method that takes a *DataContainer* and produces another *DataContainer*.
For example an *Agent* could have a DataContainer type assigned to its connection that contains a kind of XML that is used to hold the data and an associated *Entity* that has a *DataContainer* type that holds its data in a JSON format. When the Agent gives the message to the Entity to send out to its environment the Connection will used it convert to convert the XML of the Agent into the JSON format the Entity understands. As well there could be another agent that holds its data in a Prolog list like structure. As long as the Convert class implements a method that will convert the Prolog list like object to the JSON object these two different agents will have no problems communicating with the entity that send JSON out to the entity it is connected to I the external environment.

1.1.2 DataContainers

DataContainers are used to hold the message information that is received and sent to the external environment that the Agents and Entitys are connected to. The *DataContainers* can hold the data in whatever structure they like but they do currently need a method called getSource() that converts that data into a *string* to be sent out the connections to the external environments. As well to decode the information received from the external environment currently strings are used because the example being used uses TCP connections however that Connection class used can overridden to use another method of decoding the information received from the external environments.

1.1.3 Implementing a new message standard

The Interface is designed to make it easy to implement a new message type to help the system adpat to different environments easily.

1. Create a new DataContainer class. It is possible to extend from an already created class, this would be advised if there is already a DataContainer type that contains data similar to the data that is desired to be used. For example if XML data is being used the class XMLPercept is designed to hold XML data and can convert to a couple different DataContainer types.
2. After a new class is created the convert method will need to be implemented. The implementation of this method will make it possible for this message type to be used with other message type by being able to convert between them. Now, all

data description langauges are not created equal so some information might be lost in translation but this is unavoidable because the translation is nessecary to communicate with other environments. This framework makes it possible for every Agent and Entity to speak a different language and still understand each other as long as a proper convert method is written for each other Agent and Entitys message type. Look in the UnrealInterface project for examples of these convert methods.

1.2 Connections

Connections are used to connect the Agent or Entity to an external environment. In the examples used byte arrays are sent between the EnvironmentInterface and external interfaces. On the lowest level that is byte arrays are being sent back and forth. When a message is received by a Connection the full and complete message is given to the DataContainer class to decode and store inside of the *DataContainer* class.

A Connection is intended to have a Convert class and a *DataContainer* class assigned to it. The *DataContainer* class is used to decode and encapsulate incoming messages and the Converter is used to convert *DataContainer* types given to it to the *DataContainer* type that is assigned to the Connection.

1.2.1 Incoming messages

In the TCP example used when a message is received the Connection buffers until it is sure it has received a full message. After a full message is received the message is given to the assigned *DataContainer* class to decode and encapsulate. **The DataContainer will then be passed to the listeners of the Connection.**

1.2.2 Blocking on messages

The Connection Classes with have their own treads to support the incoming message receiving. Message receiving will most likely have a blocking method call in it to wait for a message to be received.

In the UnrealInterface example The Agents and the Entitys use TCPNIOConnections wich has a blocking method getMessage() which blocks on a read() on the underlying TCPConnection. The thread should always be listening for an incoming message and when it is received the designer of the Connection decides how it will be decoded but in this case a full message is buffered and the message is given to the *DataContainer* class to decode.

Talk about the threading and the message receiving.

1.2.3 Outgoing Messages

When a message is passed to the listeners of a Connection this is done using the NotifyListeners() method in the Connection class. This method will pass the *DataContainer* to the handleData() method of the handler that currently is designed to pass the DataContainer to all of the Agents or Entitys that are associated with the Entity or Agent that is assigned to the Handler. Sending the message will involve the use of the Convert and will attempt to convert the message before it is sent to the external environment.

1.2.4 Creating a new Connection type

Without re-writing one side of the envirnoment there are a few ways to create a new Connection type. The easiest way is to go along with the EnvirnomentInterface and extend the aiInterface.connection.Connection class. Which supports the Interfacer interface that on the base level sends arrays of bytes. For example if JNI were desired, the information for the message could be packed in the array in a string form and passed to the other native environment. The Interface interface also support sending DataContainers, this could be advantagous if one wanted to use J2EE or send serialized data in some maner.

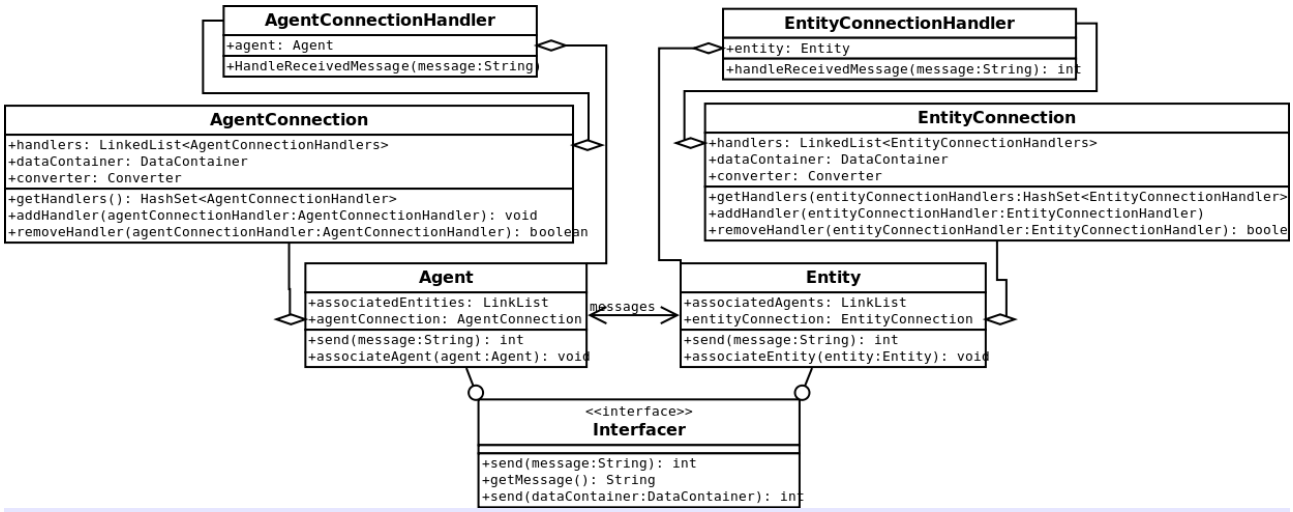
1.3 Agent-Entity Associations

The interface is designed to connect multi-agent systems in different environments. The interface is designed to do its best to not restrict the type of environments that are connected to. TCP has been used and so has UDP but as well JNI ect could be used to connect environments.

Agents and Entitys are associated with eachother using the primary client class *EnvironmentInterface*. The Agent and Entitys classes primarily only differentiate for naming conventions. They have the same methods that are used by the *EnvironmentInterface* to associate each other. Each Agent and Entity has a LinkedList of Entitys if it is a Agent or Agents if it is a Entity. These lists will be used to access the connections and then the Converting classes and DataContainers so that messages can be sent out to all of the associated Entitys or Agents that are connected to their external environments.

Figure 1.3.1

This is a diagram discribing the associations that provide the multi-agent and possibly even multi-environment interface.



Need to update the Handlers to include the handleData() method.

1.4 UnrealInterface Example

1.4.1 Introduction

1.4.2 Connections

The UnrealInterface uses TCPNIO type connections. TCPNIO is a newer faster type of TCP connection that can communicate with regular TCP connections without any issues. See the TCPNIOConnection class for reference to the code in this report.

1.4.3

1.4.4

1.5 Agent Behavior

1.6 Agent Behavior

1.7 Agent Behavior

1.8 Agent Behavior

1.9 Agent Behavior