Bluetooth Host Controller Interface (HCI) using the Unified Modeling Language Ô (UML Ô)

by Andrew Lai, Goi Sio Peng, Heng Kwee Tong, Zhang Guohui and Huang Bin
Institute for Communications Research, Singapore

Introduction

The Unified Modeling Language™ (UML™) represents an important step in the evolution of programming languages. Compared with the more traditional text based programming languages such as C, Pascal, Fortran and Java, the Unified Modeling Language is in one sense more user friendly because of the various diagrams and charts that are associated with it (e.g. Object Diagram, State Chart, Message Sequence Chart). Using the various diagrams and charts, even the uninitiated programmer, can quickly learn the essential flows and functions of the code.

A team of engineers from the Institute for Communications Research, Singapore, has been experimenting with the I-Logix™ implementation of UML, Rhapsody®. In particular, the team has been developing part of the Bluetooth protocol stack using Rhapsody. At the time of this writing, the Bluetooth HCI and L2CAP Layers have been completed and tested. All work was done in the Windows 2000 environment.

Bluetooth in a Nutshell

Bluetooth wireless technology is a specification designed to enable wireless communication between small mobile devices. The inspiration behind this technology was to replace the cables currently required to enable device connectivity. Imagine a scenario in which both the laptop PC and the digital camera use Bluetooth wireless technology. In this case, there is no need for cables to transfer data between the devices. Expanding this idea to include all hand held mobile electronic devices is, in a nutshell, the Bluetooth wireless technology vision.

In addition to eliminating the need for cables and dongles to connect devices, Bluetooth enables devices to form small, ad hoc wireless networks called piconets. Bluetooth communication uses the unlicensed ISM band at 2.4GHz. The transceiver utilizes frequency hopping to reduce interference and to combat fading. A typical Bluetooth device has a range of about 10 meters. The communication channel can support both data (asynchronous) and voice (synchronous) communications with a total data rate of 1 Mbit/sec.

Figure 1 shows the complete Bluetooth protocol stack. The protocol had been designed to include existing protocols such as TCP, UDP and OBEX. In addition, the Bluetooth specification also defines a Host Controller Interface (HCI), which provides a command interface to the Baseband controller, link manager and access to hardware status and control registers, which is the essence of what this project is about.

I-Logix Inc. • 3 Riverside Drive • Andover , MA 01810 • Tel: 978-682-2100 • Fax: 978-682-5995 • http://www.ilogix.com

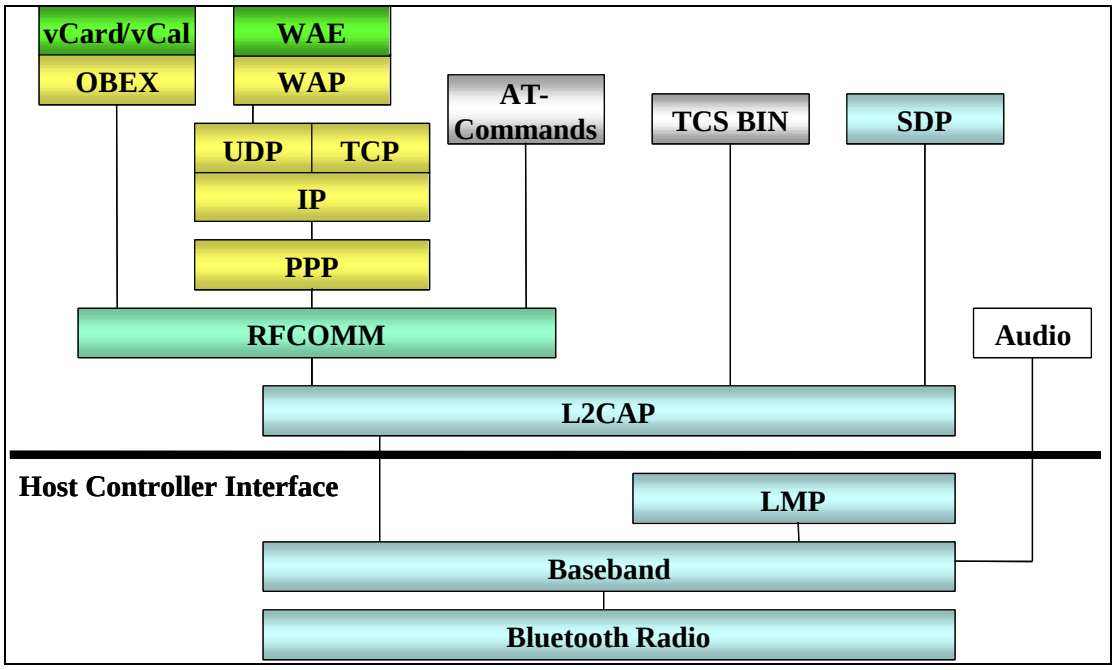

Figure 1 Bluetooth Architecture

Modeling the HCI Layer

Figure 2 below shows the Object Diagram of the HCI Layer. The first challenge was to identify the classes that are needed to describe the *HCI_Layer*. The actors shown below, interface to the HCI Layer through their relationships with the *HCI_Layer*, and essentially call upon the operations within the *HCI_layer*. Using these actors, test cases for verifying the *HCI_Layer* were performed.

I-Logix Inc. • 3 Riverside Drive • Andover , MA 01810 • Tel: 978-682-2100 • Fax: 978-682-5995 • http://www.ilogix.com

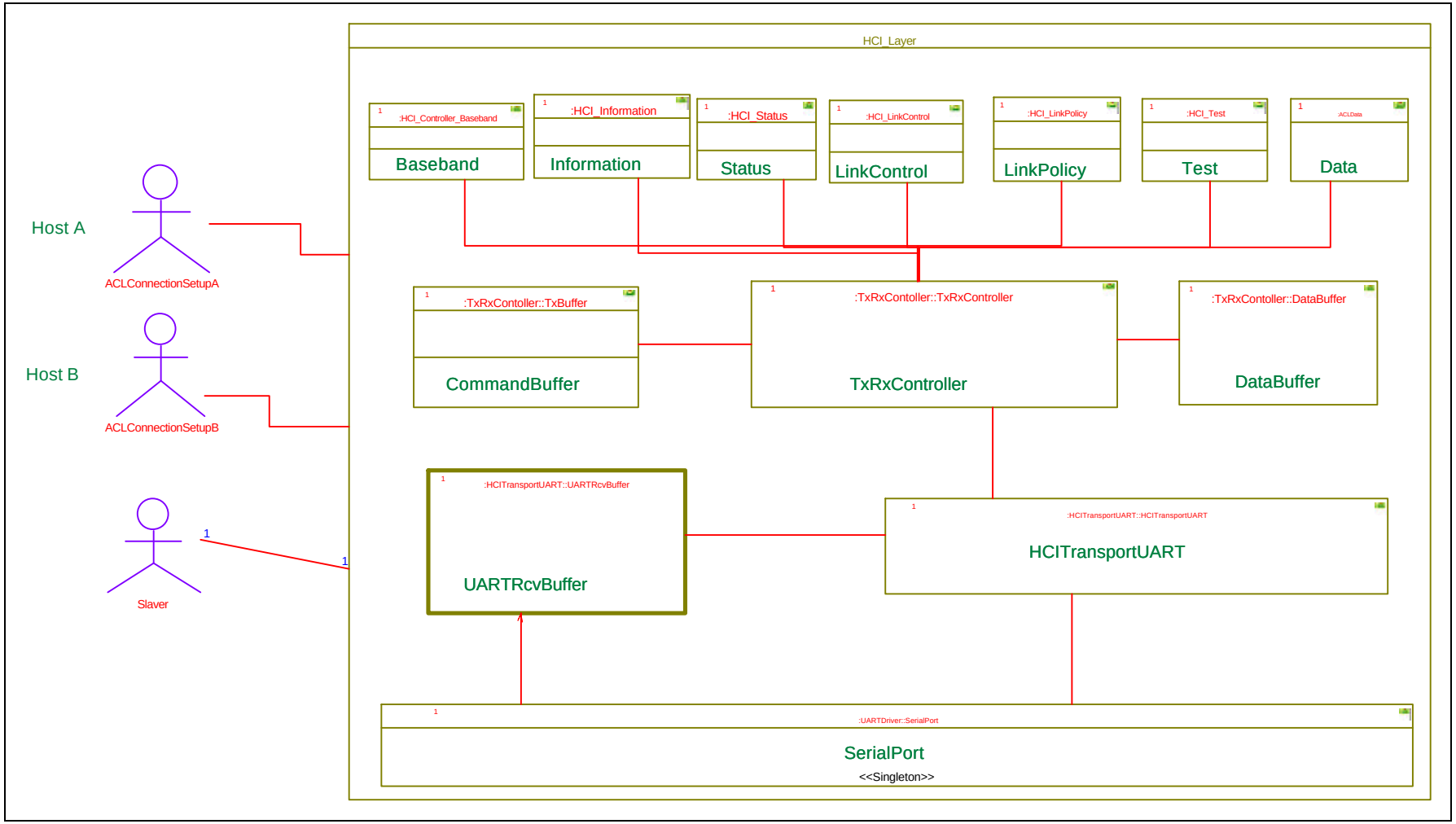

Figure 2 Object Diagram Representing the HCI Layer

I-Logix Inc. • 3 Riverside Drive • Andover , MA 01810 • Tel: 978-682-2100 • Fax: 978-682-5995 • http://www.ilogix.com

It was decided at the beginning of the project that the design of the UML model of the Bluetooth stack should be in such a manner that is easily understood by anyone reading the Bluetooth Specification. This would have been easier to do if the Bluetooth Specification were written in an object-oriented manner. However, since this was not the case, certain object-orientation had to be imposed over the Bluetooth stack. After going through the Bluetooth HCI Specification, it was decided that 6 objects corresponding to the following sections of the HCI Specification (Ver 1.1) would be created :

1. Link Control Commands
2. Link Policy Commands
3. Host Controller and Baseband Commands
4. Informational Parameters
5. Status Parameters
6. Testing Parameters

These objects will interface through relationships, to a *TxRxController* Object which will take care of all Command, Event and Data packets. The *TxRxController* in turn interfaces to the HCI Transport Layer below it (called the *HCITransportUART* since we were using a UART interface). The *HCITransportUART* will in turn interface with the *SerialPort* which communicates with the serial port through Windows System calls.

The *TxRxController* connects to a *DataBuffer* and a *CommandBuffer*. The purpose of these buffers is simply to enable the HCI Layer to receive data and commands faster than they can be sent out through the UART. Since the *UARTRcvBuffer* already buffers all incoming packets, there is no need to buffer the receiving packets again in the *TxRxController*.

At the lowest layer of the HCI, is the *SerialPort* object, which interfaces directly to the serial port of the PC. The point of having this object was to enable an easier upgrade to USB enabled stack in future. The *SerialPort* object uses system calls to get data in and out of the PC's serial port. This *SerialPort* object is configured to do a non-blocking read from the PC's serial port.

The work was divided among a few engineers. Each engineer would essentially be responsible for one or a few objects. The objects were distributed across packages so that the packages could be integrated. This feature allows classes, events and operations, programmed by someone else, to be added to the model.

Statechart for HCI Controller

The six groups of objects, namely Link Control Commands, Link Policy Commands, Host Controller and Baseband Commands, Informational Parameters, Status Parameters and Testing Parameters, have similar state charts as shown in Figure 3 below. Each object is driven from one state to another through events. All event names start with the letters "*ev*".

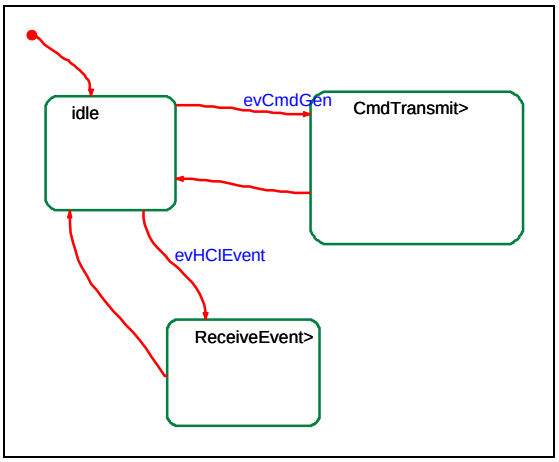

Figure 3 State Chart for the Six Objects

Each object has two states, a *CmdTransmit* and a *ReceiveEvent* state. The *CmdTransmit* state will generate an event to the *TxRxController* while the *ReceiveEvent* will process and print out the returned parameters. The individual commands are provided as operations in each of the objects.

The *TxRxController* state chart as shown in Figure 4, is a more complex one in that there are five concurrent subsystems to handle the flow of commands and data and the reception of events. Commands and data are queued in the *TxBuffer* and *DataBuffer* respectively before being transmitted out. This concept of concurrent events is important as it makes the model look cleaner, and is more readable.

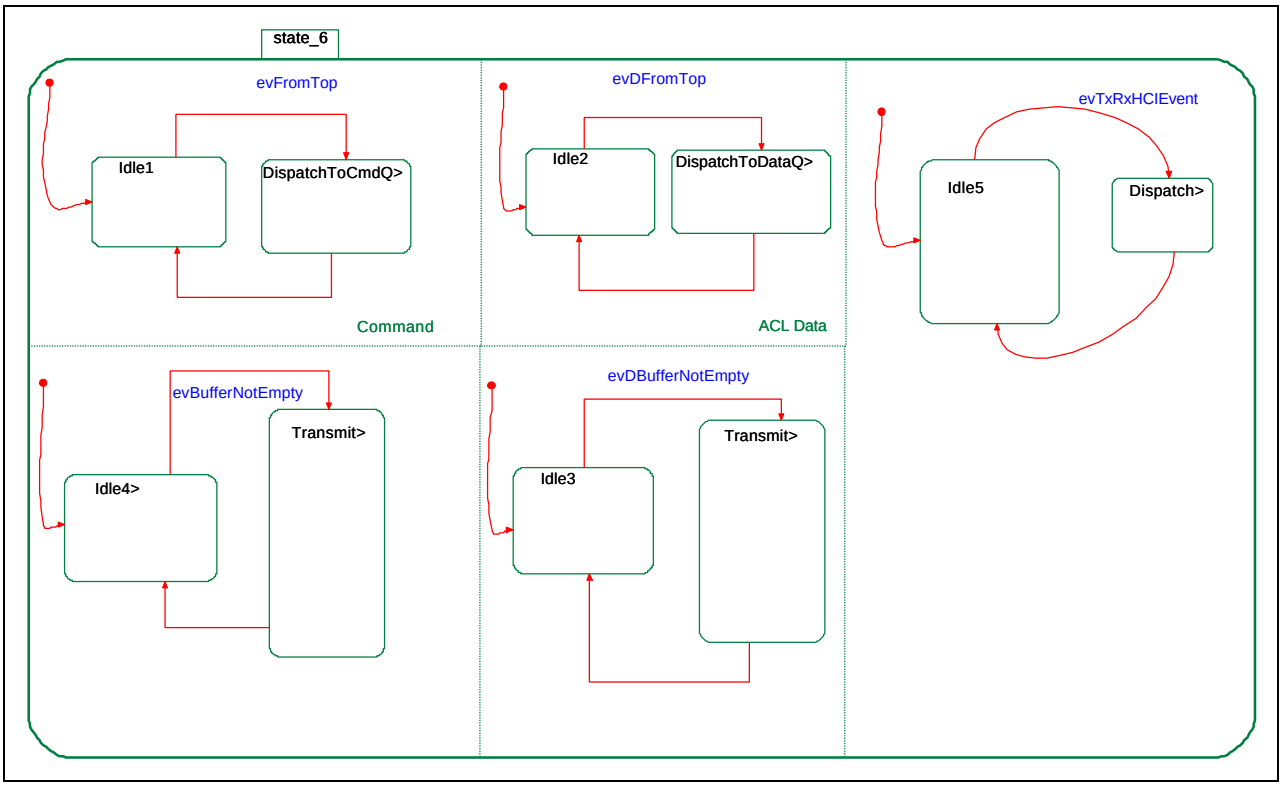

Figure 4 State chart for TxRxController

In the *HCITransportUART* state chart (see Figure 5), the transmit state appends the HCI packet indicator before sending HCI packet to the *SerialPort* driver while the Receive state gets data from the *SerialPort* driver before passing it to the *TxRxController*.

I-Logix Inc. • 3 Riverside Drive • Andover , MA 01810 • Tel: 978-682-2100 • Fax: 978-682-5995 • http://www.ilogix.com

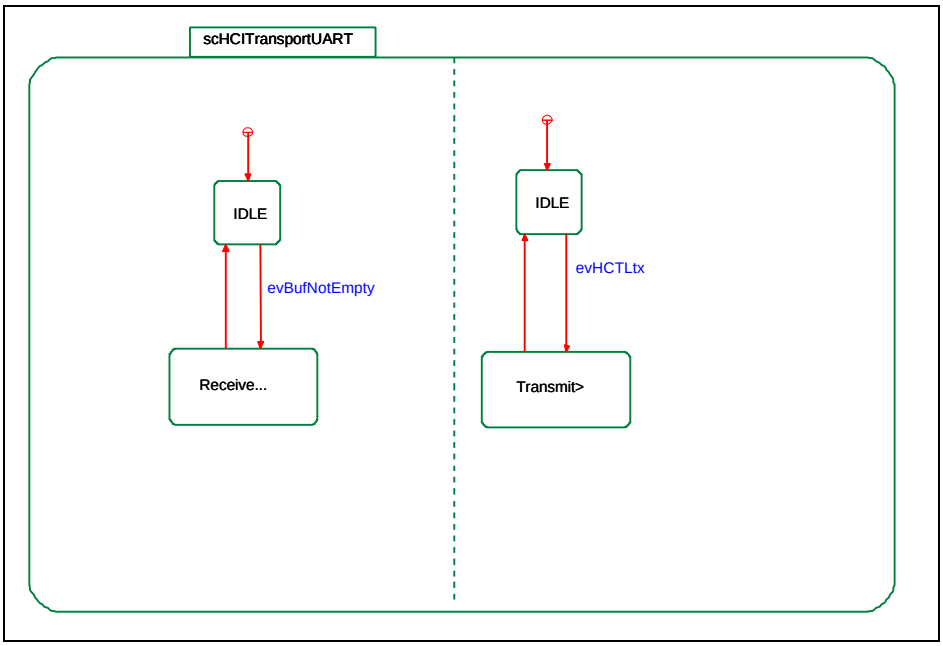

Figure 5 State Chart for HCITransportUART

Lastly, the *SerialPort* state chart (see Figure 6) provides the system calls for interfacing with the serial port.

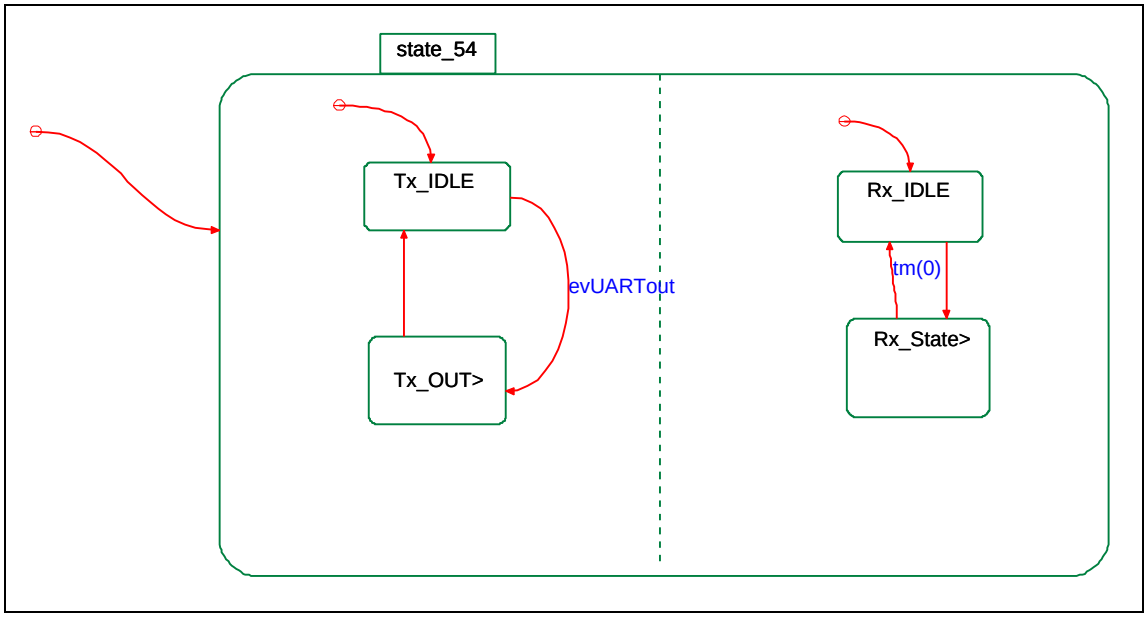

Figure 6 State Chart for SerialPort

Testing the HCI

The UML model was tested by using Ericsson's Bluetooth Hardware Modules. Basically, two HCI-enabled Bluetooth devices, each connected to a personal computer running Windows were used. This set up is shown in Figure 7 below.

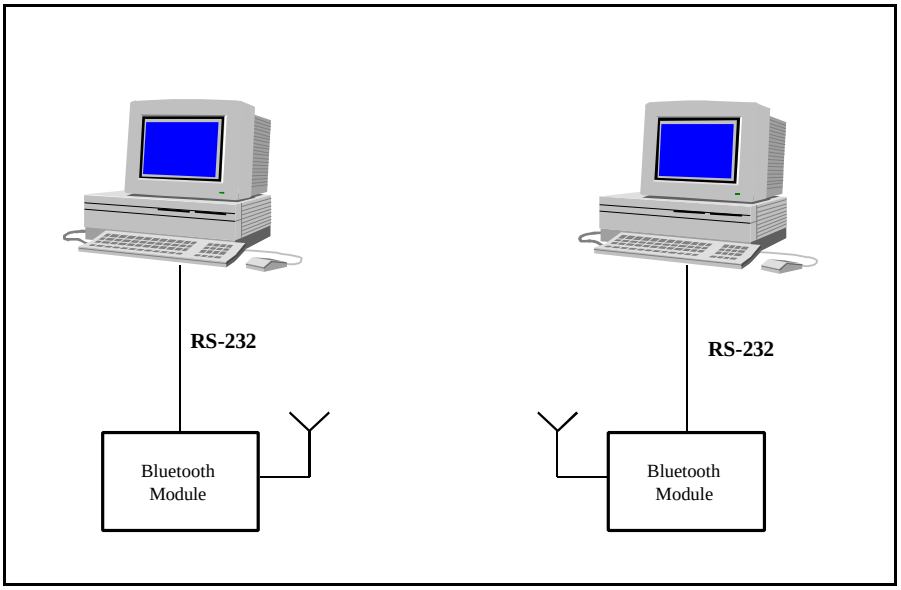

Figure 7 Bluetooth Hardware setup

Then the following procedure was carried out :

1. Computer A initiates a connection using the HCI Commands.
2. Computer B receives the connection request and accepts the request using the HCI Commands.
3. After the connection is set up, data is transmitted and received, between the two computers, A and B.

To ensure that the model performs as expected, a Message Sequence Diagram is produced, and inspected. The Message Sequence Diagram shows the sequence of program threads from one object to another. The objects shown here are the *HCI_LinkControl*, *TxRxController*, *DataBuffer* and *TxBuffer*. The Message Sequence Chart in this instance shows what is happening when one Bluetooth module is making a connection to another. *HCI_LinkControl* triggers an event *evFromTop* in the *TxRxController*. The *evFromTop* event carries with it the HCI Command Packet. The *TxRxController* places this packet in the *TxBuffer* by triggering a similar event, *evFromTop* in the *TxBuffer*.

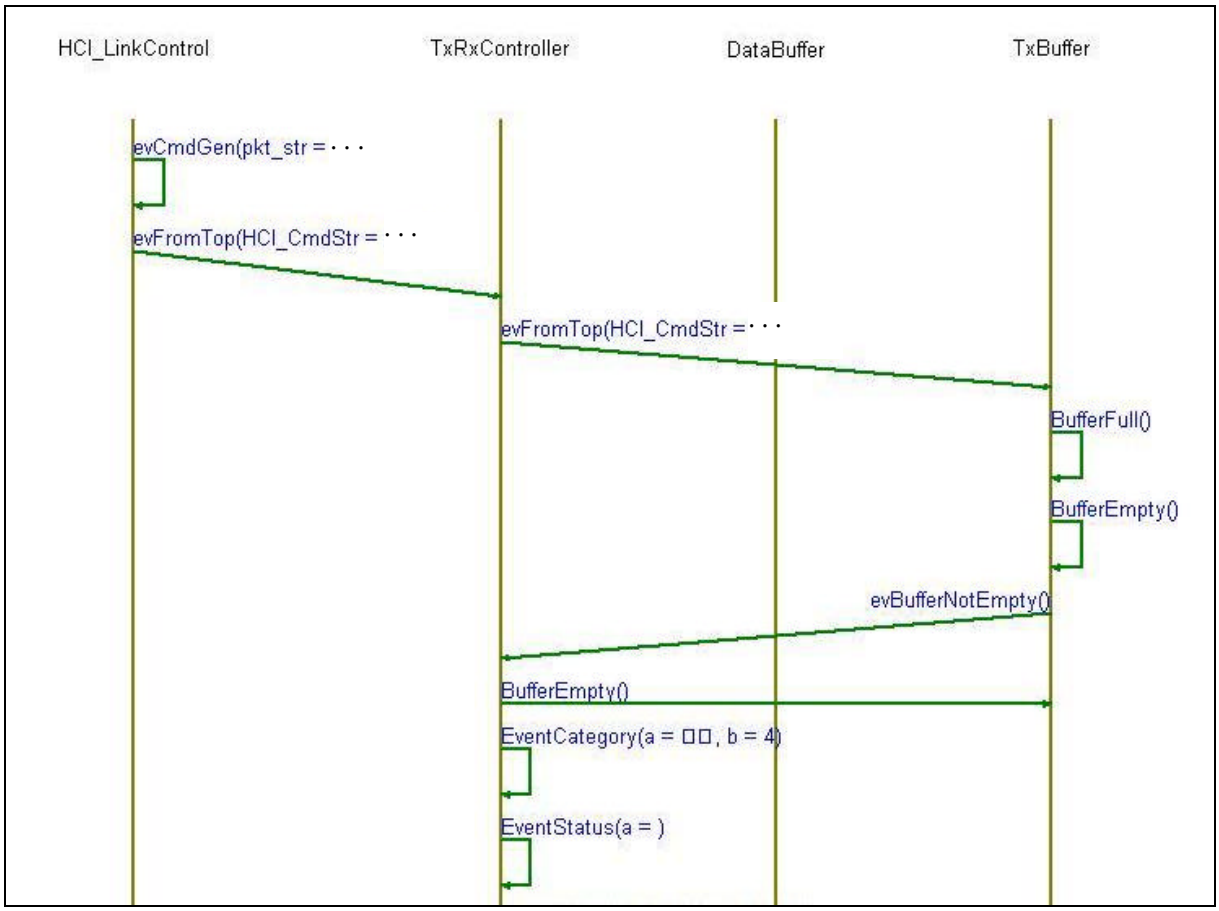

Figure 8 Message Sequence Chart during Connection Setup

Linking to a Graphical User Interface(GUI)

After the software is written using UML modeling language, it is linked to a GUI written in Visual C++ using Microsoft Foundation Classes (MFC). Rhapsody allows programs developed in another environment and language to use the binaries it generates.

To enable the model to be used in a GUI developed using Visual C++ MFC classes, it is first necessary to generate a library in Rhapsody. This is easily done by changing the options in the Component Directory, to effect a change of the component to a library instead of an executable. Once a .lib file is produced, this can be linked in the usual fashion by declaring the .lib file in the Visual C++ development environment. The figure below shows the appearance of the GUI on the Computer screen. The user can either type in the required HCI Command in the "Command Input" edit box, or select them using the "Command Groups" and "Select Command" combo boxes. "Command Groups" allows the user to narrow down the selection to a group of commands as specified in the Bluetooth HCI Specification. "Select Command" then allows the user to select a particular command.

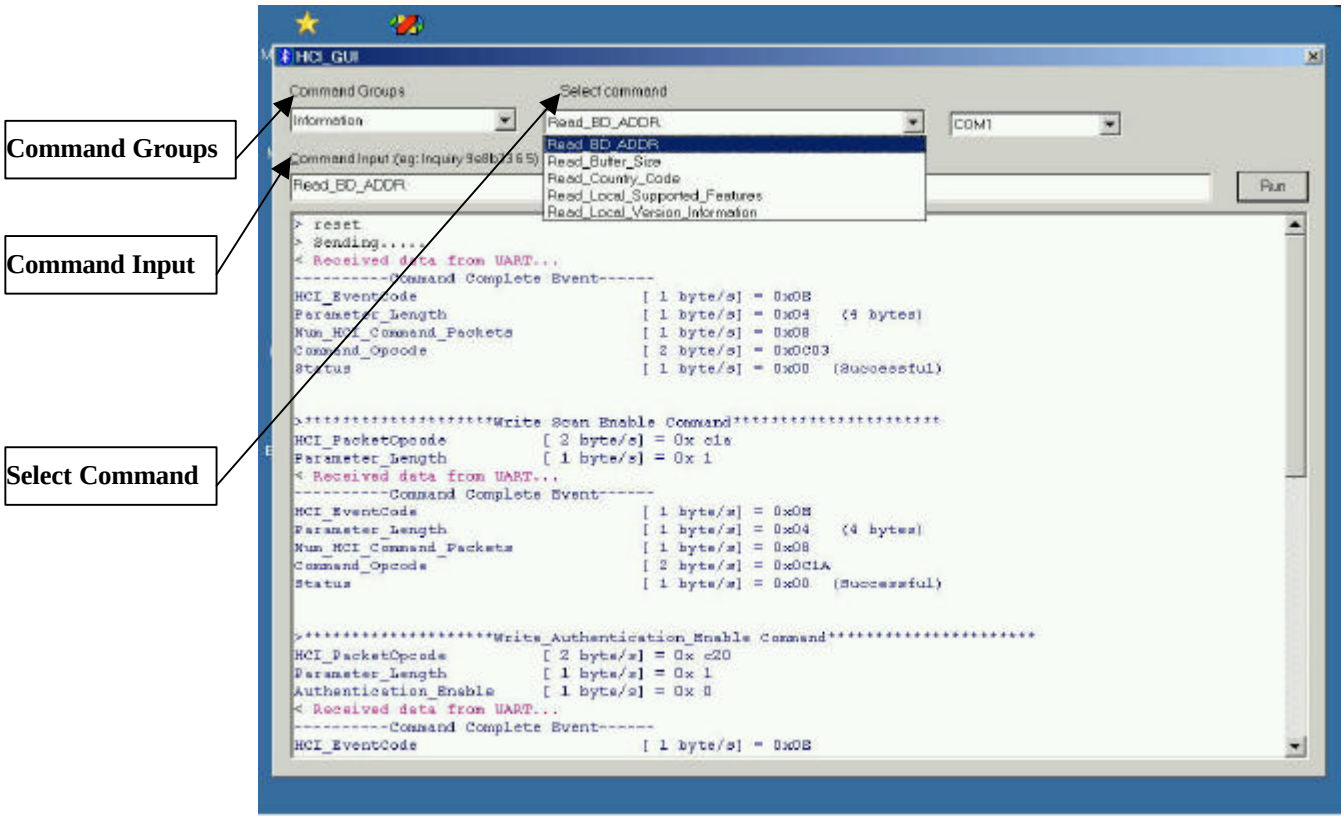

Figure 9 Graphical User Interface designed using Visual C++, MFC

Conclusion

The main challenge in the development of software using UML, is identifying the objects. In a short span of three months, two engineers, relatively fresh from University, were able to successfully code and test the Bluetooth HCI layer. Although there is a learning curve to overcome when using UML, the end result was much easier to understand than direct hand coding in text files. By looking at the various diagrams (e.g. Object Diagram, State Chart, Message Sequence Chart), it is possible to quickly understand what the programmer is trying to achieve. UML is definitely the way of the future.

For further enquiries, please email: laiandrew@cwc.nus.edu.sg

References :-

1. Bluetooth Specification Version 1.1 Core, http://www.bluetooth.com/pdf/Bluetooth_11_Specifications_Book.pdf
2. Jennifer Bray and Charles F Sturman, Bluetooth - Connect without Cables, Prentice Hall 2001.
3. Grady Booch, James Rumbaugh, Ivar Jacobson, The Unified Modeling Language User Guide, Addison Wesley 1999.

Disclaimer :
The article shares the users' experience with I-Logix Rhapsody UML Tool and in no way constitutes to an endorsement of the product.

I-Logix Inc. • 3 Riverside Drive • Andover , MA 01810 • Tel: 978-682-2100 • Fax: 978-682-5995 • http://www.ilogix.com