Using leJOS communication for games

Timo Paukku Dinnesen (timo@daimi.au.dk)
University of Aarhus
Aabogade 34, 8200 Aarhus N, Denmark

June 3, 2005

Contents

1 Introduction

A popular choice for a software development kit(SDK) for the LEGO Mindstorms RCX is leJOS[leJOS]. The leJOS SDK supports writing programs in JAVA that can be run on a virtual machine on the RCX. Since JAVA is a high level language it enables the user to do advanced things like communication without knowing much about the technical aspects in the actual transmission. The leJOS API specifies several different ways to do infrared(IR) communication between the RCX and the IR tower. This appendix describes some of these and their properties with the focus being on using leJOS for implementing robot games.

2 Game communication

Since the properties that define good communication are very different depending on which game and which goals are intended we here chose to look at games with the following properties.

1. The game will involve more than one robot.
2. The game will be played by human players. This means that one or several humans will control one or more robots. We are not looking at robot only games where the challenging part is coding the robot in advance.
3. The player is in direct control with his robot. This means that if the player leaves the controls his robot will not do much on it's own. The player will need to direct it's every move.
4. The game can also include robots that are only controlled by programs. These programs may be run on the robots themselves or may be run on a PC that then controls the robot by communicating with the robot.
5. The player is not required to know a lot about robotics or the game's code or ways of working. For example we can't assume that the player will know how to restart or perform maintenance on the game if it breaks down.

For a game like this to work the communication part is likely to need the following properties:

- **Communication in two directions.** Since the player will control his robot directly the communication will need to support some sort of commands given from the player to the robot. The other direction is needed to report when the game is over, perhaps if some loose condition has been meet by the players robot.
- **Fast communication of remote control messages.** The player will not feel "in control" if his robot responds to commands in a very slow way. If the game requires precision this property becomes even more important.
- **Stability.** If the communication can break down the player might not notice this and wonder what he is doing wrong. Also it is a know issue that errors in games makes them less fun. For example it's not fun not to be able to play a game because it simply won't start or function correctly.

The following properties are not that important:

- **Guaranteed delivery of all packets.** In a game the "state", as in "what is happening right now", is changing constantly. So if a packet is lost it might not matter much because the information in it might be obsolete anyway a very short time later.

- **Addressing of packets.** If information is broadcasted to all robots they may all be able to use the information for something useful. This will not be possible if a lower layer in the protocol has already deleted the packet because it was not specifically targeted for the robot. Addressing can always be implemented at higher layers where it is needed if the communication method delivers the data in packets.

3 The communication classes in leJOS

The communications methods in leJOS are all, except the Serial class, part of the RCX-COMM package. RCXCOMM started out as a project done by the LEGO3 Team at DTU-IAU[LEGO3]. It has since been improved by different people to support the USB IR Tower and several new protocols.

Unfortunately many of these improvements are only for use between with a single RCX and a single IR-Tower. This is probably because the LEGO3 Team chose to use acknowledge packets(ACK) to reply to messages to ensure that data get though. While this is perfect for the use in their project it does not work when more than two communicating parties try send each other messages.

Since the messages usually aren't addressed the sending of ACK replies leads to problems. Since the tower may receive many ACK's to a single broadcast it can never be completely sure about if the packets actually got through. On the RCX side a lot of messages might have been lost but ACK's to the messages might have been sent from other RCX's, so the end result is often that the RCX's are still waiting for more data, but the tower does not send any because it thinks that all has been delivered properly.

4 The Tower class

On the PC side the Tower class is the basis of all communication. All the RCXCOMM classes use the Tower class when they are running on the PC. And if one needs to communicate with the Serial class the Tower methods sendPacket() and receivePacket() must be used since the Serial class does not exist on the PC side. The same for the LLC class. Here the low level methods read() and write() can be used to send raw bytes.

5 RCX IR hardware limitations

The RCX IR messages are transmitted using a 2400 baud signal rate. Since the RCX does not use *quadrature amplitude modulation*[1] or other data compression techniques the 2400 baud equals a transmission speed of 2400 bits/sec. This equals a theoretical[2] limit of:

$$\frac{2400 bits/sec.}{8 bits/byte} = \underline{\underline{300 bytes/sec.}}$$

6 The test setup

To test the practical use of the different communication methods a remote control program was created for the PC. The program was designed with the intension that the remote controlled robot should receive commands at a steady state. These commands should hold a complete state of what the robot is supposed to do. For example a command could be "Go forward and turn left", and the robot should then

[1] A commonly used technique used to encode more than one bit of information into each bit. For example a 2400bps modem might transmit at 600 baud. For a complete explanation see [Stallings, 00, p. 161] & [WikiPediaQAM]

[2] Since the data transmission also uses some bits for startbits, stopbits, et.c. the value is unrealistically high.

do both of these at the same time. When the user does not do anything a command should say “Do nothing” and the robot should stop all motors.

Since some of the protocols do not guarantee delivery of data, the commands need to be sent more than once to increase chances that some of them arrive. Lost data does not matter much if the program is repeatedly sending commands since the robot might pick up a new command (perhaps holding the same information) a short time later. If the commands are sent often the user probably never notices that a command was lost.

A diagram of the testprogram can be seen in figure 1 on the next page.

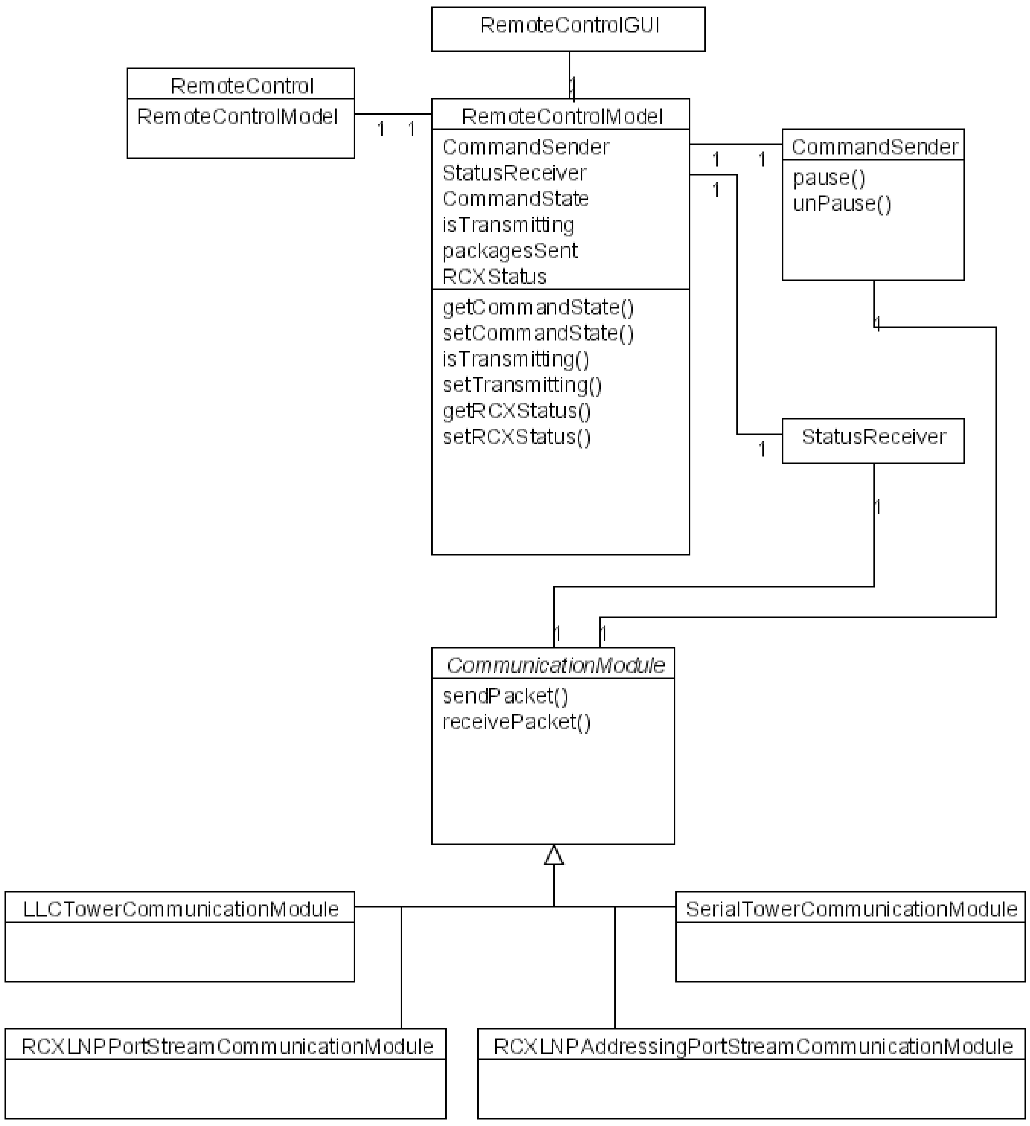

Figure 1: JAVA Class diagram of the test program.

The different classes can be described as:

6.1 RemoteControl

A simple main class that spawns a new RemoteControlModel. It also creates the communication module specified in the arguments passed via the commandline, but does not use the communication module for anything, it just passes it on to the RemoteControlModel.

6.2 RemoteControlModel

The model that contains the core of the remote control. It holds a few variables and implements getters and setters for these.

- **commandstate** holds the command that should currently be passed onto the robot.
- **isTransmitting** a flag that shows when the IR port is being used. It is set from the CommandSender and is only used by the GUI for debugging purposes.
- **packagesSendt** A counter counting the number of packets that have been sent. Placed here so the GUI can access it. It is used to check the transmission quality. (A similar counter is placed in the robot for recording how many packages are received.)
- **RCXStatus** A number representing the status of the RCX. In this simple test it does not really have a purpose, but it is set when status messages are sent from the RCX to the PC.

6.3 CommandSender

A thread that runs at a specified interval to send commands to the robot. If the interval is set lower than the communcation module can handle the CommandSender will try to send as often as possible. CommandSender contains two methods, pause() and unPause(), that can be used to pause the sending of messages.

6.4 StatusReceiver

A thread that runs at a specified interval to request status messages from the robot, and handles reception of these. For some of the low level communication modules the StatusReceiver use the pause methods in CommandSender to stop sending of commands while listening for incoming status messages.

6.5 CommunicationModule

CommunicationModule is an abstract class that specifies that a communication module should contain methods for sending and receiving messages.

6.6 LLCTowerCommunicationModule

This communication module uses the Tower class(Which is a part of RCXCOMM) to send messages intended for reception by a RCX program that uses LLC for IR communication.

6.7 LNPPortStream-CommunicationModule

This module uses LNPPorts supplied by RCXCOMM to do communication with a RCX that also uses LNPPorts.

6.8 LNPAddressingPortStream-CommunicationModule

This module uses LNPAddressingPorts supplied by RCXCOMM to do communication with a RCX that also uses LNPAddressingPorts. An LNPAddressingPort needs to have a hostnumber and a portnumber set. For the purpose of this test these numbers were hard coded into the program.

6.9 SerialTower-CommunicationModule

This module uses the Tower class to send messages intended for reception by a RCX program that is using the Serial class. This is done by wrapping the data into a packet with an opcode for datatransfer.

7 The second test setup

Since the use of streams in some of the communication methods does not really need the two classes CommandSender and StatusReceiver a simpler version of the test program was written. The results from these tests are added to the results where they are appropriate.

8 leJOS communication

This section will describe the different leJOS communication classes and for some of them describe how good they will perform in robot games with the properties mentioned above.

8.1 RCXPort

8.1.1 Description

The RCXport is the result from the LEGO3 Team's work with RCXCOMM. It supports the user with JAVA input and output streams and thus enables easy communication. It ensures that data get though by buffering the data if the two communicating parties are not within range. It does not support addressing.

8.1.2 Usability in games

Because the protocol ensures that data get through is was not tested, as this way of communicating increases overhead in packets and is not needed for game purposes. Also the problem mentioned above with ACK's happens with the use of more than one RCX.

8.2 RCXLNPPort

8.2.1 Description

The RCXLNPPort also supports the user with JAVA streams, but this version uses the Legos Network protocol(TODO: find link). It's does not ensure that packets get through but ensures that packets that get through are not corrupted.

8.2.2 Transmission speed

Using the RCXLNPPort the remote control program was able to send 5 one byte commands per second to the car. This gives a response time of 200msec. This is not very fast compared to computer games, but might be fast enough for a robot game. One good property of RCXLNPPorts is that reception of bytes from the RCX to the tower is not have a big effect on the rate at which commands can be sent to the RCX. When using the second test setup the transmission speed was increased to 8 one byte commands per second. Since RCXLNPPort uses packets with a length of 4 when sending one byte (Header, length, data, checksum), this amounts to 32 bytes per second. Far from the 300 theoretically possible. Of course this way of using streams is not the right one, but still the data rate seems a little low.

8.2.3 Usability in games

The use of streams makes it impossible to use more than two communicating parties. If more are used simple tests reveal that the bytes from the streams get mixed. It would not matter much if it was possible to ensure that bytes always arrived in "packets". The output streams support the methods write(byte[] b) and write(byte[] b, int off, int len) that inserts an array of bytes into the stream, and one could assume that the data would be sent as complete packets.[3] This is clearly not the idea behind streams, and simple tests also show that the bytes get mixed up like using the normal write(byte b) method. The RCXLNPPort would be fine if the "packet" used in the game was only one byte large, but it seems almost impossible to fit addressing or other advanced information into a single byte.

8.3 RCXLNPAddressingPort

8.3.1 Description

The RCXLNPAddressingPort is basically an RCXLNPPort that supports addressing. The addresses consist of a 4 bit host name and a 4 bit port number. Unfortunately it is not possible to create more than one port per host. This means that only two communicating parties can talk to each other at a time. For example a PC to receive data from two different RCX's the port would have to be opened and closed all the time.

The reason for this unusual behavior can be found in the LNPHandler. The constructor in LNPHandler creates a new Tower object. Since this can only be done once, and since every RCXLNPAddressingPort need its own LNPHandler, subsequent attempts to create more RCXLNPAddressingPorts will fail.

With some small changes to LNPHandler and RCXLNPAddressingPort it can be brought to work as intended. This destroys the fine layered structure in RCXCOMM and again shows that most of the leJOS communication methods are written for only "one tower and one RCX" uses. The changed version enables the use of one PC communicating with 15 RCX'es. Probably enough for most robot games.

[3]The documentation is not really clear on this.

8.3.2 Transmission speed

The RCXLNPAddressingPort runs at the same speed as RCXLNPPorts with about 5 one byte messages a second on the first test setup and 8 one byte messages on the second test setup. Since a RCXLNPAddressingPort uses 6 bytes to transmit one data byte, this amounts to 48 bytes per second. Communicating with more than one RCX (using the modified version), the communication slows down. With two RCX'es only 1-2 messages were received per second for each robot.

8.3.3 Usability in games

Since the addressing is not the most important feature for game communication as mentioned above the RCXLNPAddressing port may not be the perfect choice. On the other hand addressing solves the problem of mixed byte streams, and RCXLNPAddressing might be a good choice of a high level communication method when transmission speed is not important.

8.4 LLC

8.4.1 Description

The LLC class links uses native calls to some interrupt driven code running directly on the Hitachi micro controller[Caprani] that is located in the RCX. The code provides a 64 byte buffer[4] to store incoming bytes from the IR port. The methods provided in LLC are very low level. Basically only two methods are used. Read(), that reads a single byte from the inputbuffer, and sendBytes() that send a series of bytes. All packet creation and decoding is left to the programmer to create.

[4]Actually the buffer is the ROM output buffer that is re-used for input buffering. Because of this the Serial class can not be used at the same time as LLC.

8.4.2 Transmission speed

The transmission speed that can be achieved using LLC directly will depend a lot on the code written to send, receive and decode packets. Since LLC allows the programmer to create packets just as he wishes he can create packets that do not contain an equal number of 0's and 1's. This could double the transmission speed compared to the other protocols. But since the IR electronics use the number of 0's and 1's to compensate for changes in ambient light[Caprani], this might lead to lost packets.

The simple attempt used in this test did not balance the signal and did also perform quite badly. 75% of all packets was lost during transmission.

8.4.3 Usability in games

With LLC it is possible to send whatever one wants since the class is so closely connected to the IR hardware. This enables some possibilities but also requires more advanced programming to make things work.

8.5 Serial

8.5.1 Description

The serial class uses the standard RCX ROM calls (via leJOS's ROM class) to do communication. This means that no guaranteed delivery of data, no collision detection and also that the messages sent must use the opcodes that the RCX was intended to work with. To send data the user can use the method sendPacket(byte[] aBuffer, int aOffset, int aLen) and for receiving data the method readPacket(byte[] aBuffer) is available. These methods create the packet around the data so the Serial class is not as low level as LLC. For a packet to be received by Serial is must have a correct opcode set. Otherwise the RCX ROM routines will discard the packet before Serial gets it.

8.5.2 Transmission speed

The serial class is very fast compared to the earlier described classes. Using the above remote control program it was able to send 12-13 five byte commands per second[5]. Two way communication is a bit of a problem when using serial since it does not include any form of collision detection. The program has to know when data is going in what direction. For example it needs to stop sending when it knows that it is soon supposed to receive. The module written to send to the serial class uses a 17 byte long packet to send 5 data bytes including the opcode. This amounts to 204 bytes being sent per second. This places the serial class as the best attempt to get close to the theoretical maximum of 300bytes/sec.

8.5.3 Usability in games

The serial class sends data in packets which works nicely for games of the type described above. Addressing can easily be added by using a data byte as address, and still the multi casting properties can be utilized at the same time.

8.6 RCXF7Port

8.6.1 Description

The RCXF7Port is basically a RCXPort that uses the Serial class. The Serial class uses the LEGO firmware and thus only supports the official opcodes. The opcode F7 is used for data transfer in the official LEGO firmware and RCXF7Port utilizes this to enable communication with other programs that use the

[5]Actually the data part of packets were 6 bytes long, but the first byte had to be the opcode 0x05 which is one of the opcodes for data transfer.

LEGO firmware for communication. It provides streams but according to the documentation does not do it very good. Bytes may get lost and may be duplicated. It is also slower than RCXPort and it's use not recommended unless one needs the compatibility with older leJOS programs, NQC or other programs using the LEGO firmware.

8.6.2 Usability in games

The RCXF7Port uses ACK packets like the RCXPort and therefore does not support communication between several RCX's. Also when the author of the class explicitly tells one not to use it, RCXF7Port is probably the worst choice for game use.

9 Conlusion

None of the communication methods provided in leJOS are very good for games. The best choice would probably be the Serial class, but since it's very low level the programmer will need to invent a completely new protocol on top of the standard opcodes and code everything specific to the game he is going to create.

Another choice could be using LLC, but since normal use of the class would involve headers and balanced 0's and 1's the result would be so close to the packets sent by the Serial class that it seems like unnecessary extra work to do so.

None of these solutions are hiding many of the technical aspects from the user, so the hope of using a high level language for easy implementation of game communication should not be reason for choosing leJOS.

References

[Caprani] Ole Caprani
RCX Manual
http://legolab.daimi.au.dk/DigitalControl.dir/RCX/Manual.dir/RCXManual.html

[LEGO3] http://www.iau.dtu.dk/ lego/lego3/rcxcomm/

[leJOS] http://lejos.sourceforge.net/

[RCX Internals] Kekoa Proudfoot
RCX Internals
http://graphics.stanford.edu/ kekoa/rcx/

[Stallings, 00] William Stallings
Data & Computer Communications, Prentice Hall, Inc., ISBN: 0-13-084370-9

[WikiPediaQAM] http://en.wikipedia.org/wiki/Quadrature_amplitude_modulation