Eidgenössische Technische HochschuleZürich
Swiss Federal Institute of Technology Zurich

Lior Wehrli

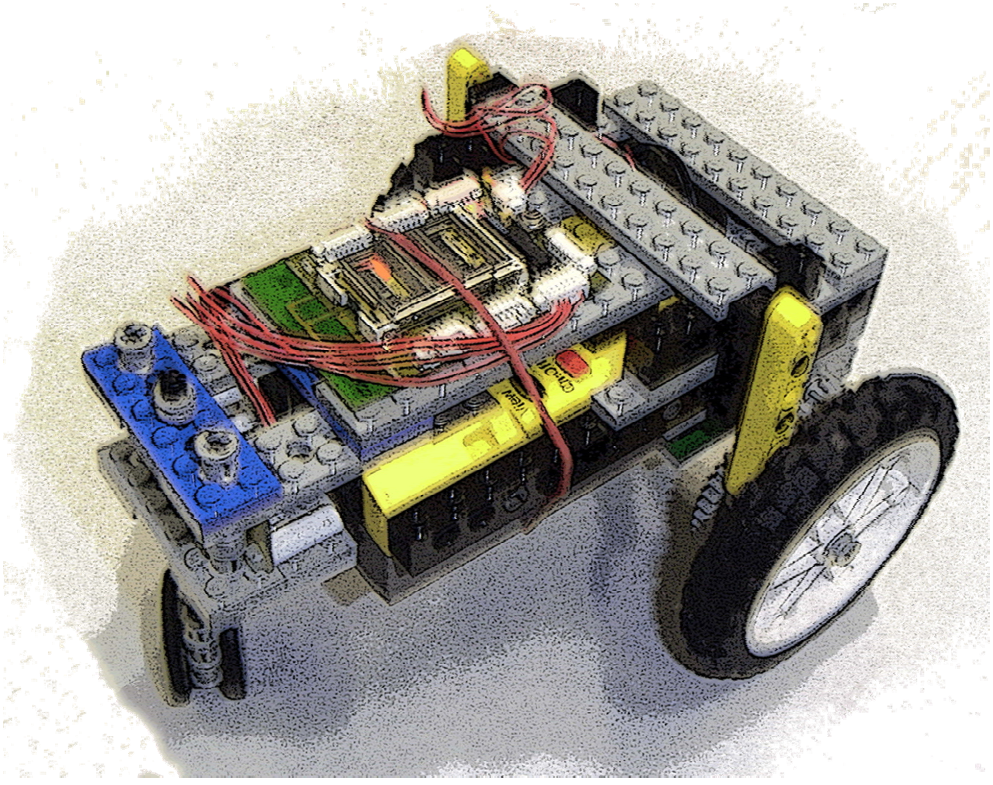
Mindstorms Admin Framework

Semester Project SA-2004-05
Winter Semester 2003/2004

Tutor: Philipp Blum

Supervisor:
Prof. Dr. Lothar Thiele

13.2.2004



Institut für Technische Informatik und Kommunikationsnetze
Computer Engineering and Networks Laboratory

Contents

1 Introduction

This chapter is a general introduction to the semester project on the Mindstorms Admin Framework. The following sections describe the task assigned to this project and the approach taken to fulfil the task. The last section gives an overview over the document itself and roughly summarizes the contents of the various chapters and sections.

1.1 Task Description

The LEGO Mindstorms Robotics Invention System [1] is currently used in many schools and institutions to introduce students to the theory of robotics and to related areas. The conjunction of the well known, simple to use mechanics of LEGO Technics and an autonomous, programmable control device inside a brick, the RCX, makes this system ideal for introductory courses [2]. The system allows students to begin with the fun part of constructing right away, without the usual planning phase needed in real robotics. The degree of complexity of the physical build up and the design of the controller can vary widely from trivial to complicated. The controller can be programmed using prefabricated modules in a simple drag and drop environment or using a full blown programming language to command the RCX.

Figure 1: Three examples of LEGO Mindstorms robots.

It is desirable to use multiple RCX bricks for complex tasks because the input and output possibilities of a single RCX are used up very quickly. As soon as the project architecture features more than three sensors for example, a single RCX brick will not suffice to process the information flow. If more than one autonomous moving machine is in use, then each has to carry at least one RCX brick for its control. But if multiple controllers are in use, then the communication facilities are of great importance since some sort of information sharing is almost always needed to coordinate the various autonomous parts of the project.

Unfortunately the LEGO Mindstorms Robotic Invention System can only establish communication between different RCX bricks by using its infra red device. The great flaw of this mechanism is that direct visible contact is mandatory from one IR device to another. For many applications this constraint means series of very hard problems as such precise navigation and adjustments as are then needed can not be expected from the system due to the relatively fuzzy nature of LEGO mechanics.

The project's first task is to solve these communication difficulties by implementing a system which, while making use of the LEGO Mindstorms Robotics Invention System, enables RCX bricks to communicate with each other regardless of their relative position - as long as they operate in each others vicinity. It is also desirable to have an interactive possibility to track information during runtime; hence the second task is the implementation

of a console application that will give students the opportunity to monitor the components and to steer various program related features on the RCX directly and conveniently from a PC terminal.

1.2 Approach

The BTnode, an all-purpose Bluetooth device developed at the ETH [3], shall be used to close the gap that opens between individual RCX bricks by routing the communicated messages over Bluetooth. A simple infra red transceiver device catches the signals sent by the RCX brick and forwards them to the BTnode, where the signals are processed and eventually sent over Bluetooth to the console application which also acts as a server for the BTnode-clients. Here the messages from the controllers are processed according to predefined or user defined rules. Commands to RCX bricks are sent via Bluetooth to the BTnodes which in turn forward them to the RCX using the IR transceiver device. Direct communication between RCX bricks is emulated by first routing signals to the console application and then forwarding them to certain BTnodes and on to the connected RCX.

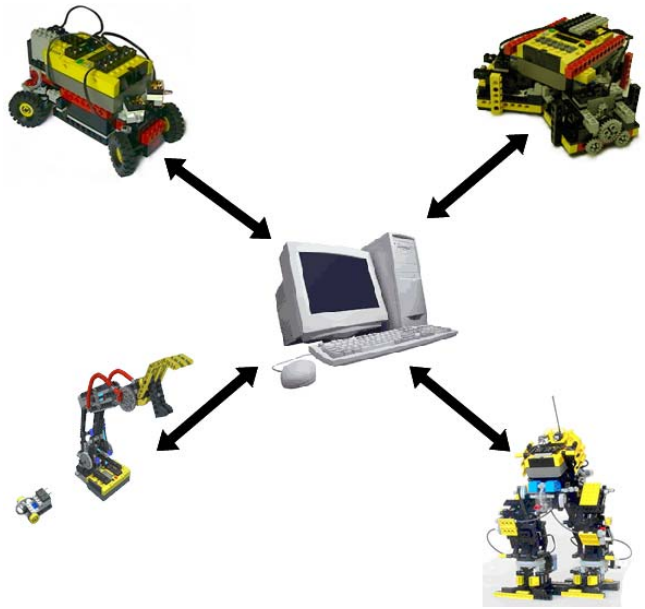


Figure 2: The PC application communicates with the connected robots and centralizes their coordination.

The PC application acts as a console towards the users and as a Bluetooth master device towards the BTnodes. Using the console, the user can also send commands to RCX modules and view the replies to the commands in a convenient form. The Bluetooth-side relies on the BlueZ [4] Bluetooth library which offers a socket interface to most known Bluetooth devices on Linux. The users can implement modules using any programming language and link them to the console using regular files or UNIX named pipes. This enables users to define their own control and coordination rules to meet with the needs of individual projects.

1.3 Overview

This section gives a short description and summary of the chapters and sections following the introductory chapter of this document. Section names are highlighted.

1.3.1 Specifications

This chapter lists the specifications of the project and defines or explains in four sections some of the technical aspects of its components. The first section lists the **Specifications** of the Mindstorms Admin Framework as relevant to its users. The section on **The BTnode** further details the functionalities and features of the BTnode device while the next section does the same for **The LEGO Mindstorms Robotic Invention System**. Both sections also mention some background information on the described systems. **The-Proof-of-Concept** is

an early implementation of some of the requirements of the project and the basis from which it started. It is introduced in the section of equal name. Finally the last section formulates a **Summary** of the chapter.

1.3.2 System Architecture

This chapter goes into some detail to discuss the internal architecture of the whole system, of its components and the way the communication is structured. The first section gives an overview over the break down of the tasks to the involved components by describing the **Distribution of Functionalities**. The following section explains how different messages are handled and routed through the **Communication Path**. The next section describes the **Interfaces** and how they are connected to the system. Again a final section gives a **Summary** over the chapter.

1.3.3 Implementation

The chapter further deepens the discussion of the presented project by detailing the internal mechanisms of the components. The sections on **The BTnode Controller** and **The Console Application** each describe how the two programs handle their tasks. A **Summary** section closes the chapter.

1.3.4 Results

This chapter discusses the results obtained from the project work. It opens with an **Example of Use** which demonstrates how a short usage by a student could look like. The section on **Stability Issues** contains a qualitative comparison of the stability of the system and of the proof-of-concept. The following section lists some **Open Issues** which were not tackled during the project or which occurred while the work was done. Again the chapter closes with a **Summary**.

1.3.5 Conclusions

The last chapter of this document discusses some qualitative conclusions made about the project results. The first section lists **Overall Conclusions** relating to Mindstorms Admin framework as a whole while the second section discusses the **Approach** in light of the finished system. The chapter close with an **Outlook** on how the system might be further developed in the future.

2 Specifications

The following chapter lists the specification of the described semester project and for clarification introduces three components that are of central importance to the project.

2.1 LEGO Mindstorms Admin Console System Specifications

2.1.1 Signals

The RCX firmware is capable of sending and receiving messages of one byte length. While there are other mechanisms for information sharing, this is the only form suitable for RCX-to-RCX communication. Because the term “messages” suggests sending of packets of variable length, or at least packets of several bytes, the term “signals” has been chosen instead for this feature. This suggests that the type of the message is the message itself and that there is no further data other then the data needed to specify the type. If the sole data byte sent is regarded as the signal type then the concept precisely fits the mechanism.

The Mindstorms Admin Framework is capable of distributing the signals sent by an RCX either to all other RCX bricks or to certain RCX bricks, according to a user given set of rules. The automatic forwarding of signals to all RCX bricks can be turned on or off by the user and the user can also independently send signals to an RCX.

2.1.2 RCX Program Management

Using the console, the user can upload compiled NQC [5] programs to any RCX by specifying the byte code file and the program slot to which the program should be uploaded to. The user can also start and stop the execution of programs on any RCX or turn an RCX brick off.

These options enable students to conveniently experiment with control programs once the physics of a machine is assembled. They can upload and test their code under realistic conditions and quickly make corrections or adjustments to the behaviour of their construction. The possibility to upload and control programs without even having to touch a robot, let alone disassemble it, clearly is a great advantage of the Mindstorms Admin Framework.

2.1.3 External Components

Distributed solutions to complex problems usually call for some sort of coordination between the components engaged in the solution. Such coordination often is central and always is specific to the approach taken to handle the problem. The console is an ideal place to implement such central coordination mechanisms. But the needs of future student projects can not be foreseen; neither should they be limited by predefined algorithms.

For those reasons the Mindstorms Admin Framework offers the possibility to redirect some information to external modules and likewise to allow external modules to control certain RCX bricks. The requirements to such modules are very low; all that is needed is a program that runs on the same OS as the console and that communicates over UNIX files using the same text interface as the user is when he or she works with the console.

The user specifies which information is to be forwarded to which modules and which modules have the permission to send commands to which RCX bricks. Allowing the user to indicate module-specific permissions enables the simultaneous use of several modules even if some are not to be trusted. Making use of the flexibility of UNIX files opens the power of

many programming languages, compiled or interpreted, for the user's utilisation according to his or her needs and preferences.

2.2 The BTnode

The BTnode is an autonomous wireless communication and computing platform based on a Bluetooth radio and a micro controller. It serves as a demonstration platform for research in mobile and ad hoc connected networks (MANETs) and distributed sensor networks. The BTnode has been jointly developed by the Computer Engineering and Networks Laboratory (TIK) and the Research Group for Distributed Systems at ETH Zurich.

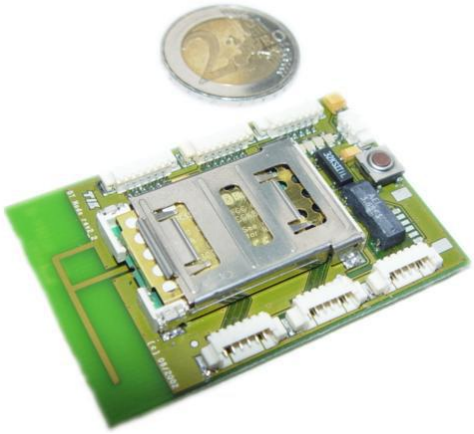


Figure 3: The BTnode

The micro controller has access to the Bluetooth device as well as to different interfaces. It is controlled by a C program which is implemented and compiled on a PC using the AVR GNU cross-compiler. The executable code can then be uploaded to the memory of the BTnode from where it will be executed by the micro controller. An extensive API allows the user to take full control over all the resources of the BTnode.

Because of the numerous connector interfaces it is possible to connect various external devices such as the infra-red device used for this project or one interface can be used for debug purposes and to display trace information.

The possibility to communicate over Bluetooth and the ease of use of the BTnode make it a perfect choice for application in combination with RCX bricks. It can also easily be plugged to the internal batteries of an RCX, so that no additional power supply is needed.

2.3 The LEGO Mindstorms Robotic Invention System

The LEGO Mindstorms Robotic Invention System is a revolutionary product that lets customers build their own robots from scratch and according to their own ideas. The kit of several hundred parts - first published by the LEGO Corporation in 1998 – includes sensors and motors that can be connected to the central component: the RCX.

The RCX is a single brick holding a micro controller, input and output connectors and a power supply in form of six AA batteries. Three input interfaces can connect to the supplied sensors. Touch sensors and light sensors are shipped with the kit but temperature sensors and rotation sensors are also available. Three output interfaces can be connected to regular LEGO motors to control their direction and rotation speed.

Figure 4: The RCX brick is the heart of the LEGO Mindstorms Robotic Invention System

A separate infra red transmitter device can be connected to a PC. This device is used to send control programs from a PC to the RCX. The user can then start the execution of one of up to five programs stored in one RCX brick. A control program can be created with proprietary, user friendly software shipped together with the

LEGO Mindstorms Robotic Invention System.

It is also possible to use some other, non proprietary software to implement more complex controller programs. Since the introduction of the product, several free source projects have developed different alternatives to LEGO's in-house software solution. Some of those alternatives include new firmware to exchange the standard firmware while others only offer new ways to implement control programs which are compatible to the original firmware.

One of the most widely used alternatives is the Not Quite C compiler (NQC), developed by Dave Baum. NQC's syntax is very similar to the C programming language but the compiled code can be executed by the standard firmware that is distributed by LEGO. This concept is ideal for students who can be expected to learn a simple programming language but who do not have the time to install and tune a tailored firmware operating system. With NQC anybody can quickly write the desired controller, even if it includes complex mechanisms, while still relying on the stability of the original firmware.

2.4 The Proof-of-Concept

The Proof-Of-Concept is a preliminary implementation of some of the specifications of the Mindstorms Admin Framework done by Philipp Blum of the Computer Engineering and Networks Laboratory of the ETH Zurich. It includes a controller program for BTnodes and an IR transmitter device assembled specifically to catch the IR signals produced by the RCX.

The BTnode controller program is based on the standard controller of the BTnode and can be compiled to either one of two execution modes: master or slave. Each BTnode's IR device is attached to an RCX's IR device. When started, the BTnodes will form a Bluetooth piconet where the node running as master will be the master node.

In slave mode the program will wait for RCX messages sent over IR and send them to the BTnode running in master mode. It will also forward messages received from the BTnode acting as master to the RCX. In master mode the BTnode distributes all incoming messages, coming from the own RCX brick or from slave nodes, to the slave nodes connected to it.

When applied in student projects the Proof-Of-Concept showed that the ability to communicate between RCX bricks is an important improvement to the original system. The possibility to exchange information and to coordinate the components of the student projects greatly improved the number of solvable tasks.

However, many new problems also showed up. The most important was the unreliability of the Proof-Of-Concept: The BTnodes would often crash or messages would go lost. Students were especially frustrated over the shier impossibility to perform a demonstration of their work without that the system would crash once or twice in an unpredictable manner. Naturally, the students wished for an improvement of the reliability.

The students also wished for a possibility to keep track of the information flow in the network of RCX and to inspect the states of the RCX during program run-time while experimenting with the robots. The requests led to the establishment of Mindstorms Admin Framework semester project.

2.5 Summary

In this chapter we have discussed the basic specification to the Mindstorms Admin Console as listed in the project requirements and as implemented in the final version. The signalling mechanism is based on the existing messaging mechanism of the RCX and allows control programs of RCX bricks to send single bytes to their piers. The distribution is in the hands of the console application and its policy can be specified by the user. The possibility to upload and control RCX programs greatly eases testing and debugging of robots by giving the

user tools to do so without having to modify the construction itself. Last but not least the interface to external modules enables students to implement their own central coordination mechanisms.

The reader has also been introduced to the most important physical components that the Mindstorms Admin Framework uses. The BTnode is a Bluetooth device developed at the ETH Zurich for experimental applications and is here used to wirelessly connect RCX bricks with a master Bluetooth device. The LEGO Mindstorms Robotic Invention System is a product of the LEGO Corporation that is widely used for introductory courses in robotics and related fields. The Proof-Of-Concept is a simple implementation of the signalling mechanism for RCX bricks using only BTnodes and simple IR transmitters.

3 System Architecture

This chapter discusses the general architecture of the Mindstorms Admin Console framework. The following sections each explain an aspect of the systems structure.

3.1 Distribution of Functionalities

There are three types of devices used in the Mindstorms Admin Console: RCX bricks, BTnodes and PCs. Each of the devices works autonomously and each is responsible for certain functionalities as detailed in the following subsections.

3.1.1 The RCX Brick

The RCX brick is the platform for the control program that steers a robot or a similar autonomous machine. The controller is not executed directly by a micro processor but is interpreted by a virtual machine. This virtual machine hides the RCX's hardware, manages the uploaded programs and offers features such as error recovery, or multitasking [6].

The controlling program reads the sensory inputs of the RCX and processes internal information. It decides on the behaviour of the machine by manipulating the direction and power level of the attached motors and it sends and receives signals to and from other RCX bricks or the console.

There is no modification to the RCX itself in the Mindstorms Admin Framework. Users can rely on the features and documentation of the RCX as supplied by the LEGO Corporation or by other sources building on top of the RCX. Students can safely ignore the Mindstorms Admin Framework when implementing controllers for RCX bricks in their projects.

3.1.2 The BTnode

The BTnode has the tasks of hooking an RCX into the registry of the console and of translating the messages and commands sent from the RCX to the console or vice versa to the appropriate protocol. The registration into the console's registry implies that the console's Bluetooth device first has to be found and a connection to it has to be established. Translating messages and commands implies that the BTnode control program parses the input from the RCX and extracts the essential information that is to be forwarded to the console and that data coming in from the console has to be coded according to the RCX communication protocol.

The BTnode control program is an indivisible part of the Mindstorms Admin Framework and should never be modified by the user. On first usage of a new BTnode the program has to be uploaded to memory but after that the BTnode is ready to work autonomously. The close connection between a BTnode and the RCX it is attached to forms a new abstract entity from the console application's perspective; a BTnode/RCX pair acting as a seemingly monolithic unit.

3.1.3 The Console Application

The console application is the centre of command for the user. It is a Linux terminal program that reads text commands from the user and displays information received from the RCX bricks. The console application also controls the Bluetooth device that acts as the master of the piconet formed by the BTnodes. It maintains connections to the BTnodes, schedules outgoing messages and analyses incoming messages. It also forwards formatted information to the connected modules as defined by the user and decides on execution or blocking of

commands coming from modules according to the permissions of the modules. It thereby hides communication related details from the user and the modules so that they can focus on the steering of the RCX bricks.

All functionalities of the console application are designed for reliability. If for example a connection to a BTnode is lost for some reason then the console application notifies the user and all modules related to the RCX. It then closes the connection and cleans up all messages scheduled to be sent to the respective RCX brick.

3.2 Communication Path

Let us follow the path of a packet of data that is sent from an RCX brick to the console application in order to demonstrate how the data is modified and where. A message sent from the RCX's IR transmitter will be caught by the BTnode's IR device. If the message is an expected reply to a previous command or if the message is a signal spontaneously sent by the program running on the RCX, then the BTnode will forward the message data to the console application.

3.2.1 BTnode-side Processing

The LEGO Mindstorms Robotic Invention System uses a proprietary packet protocol for communication with RCX bricks. However, the connection between a BTnode and the terminal application is a Bluetooth L2CAP connection. This means that the communication is fault free, that the packet sequence is preserved and that the packet borders can also be preserved. Therefore there is no need for the packet handling features of the RCX communication protocol with its flow control mechanisms. In fact, there is no need for any of the features of the RCX communication protocol.

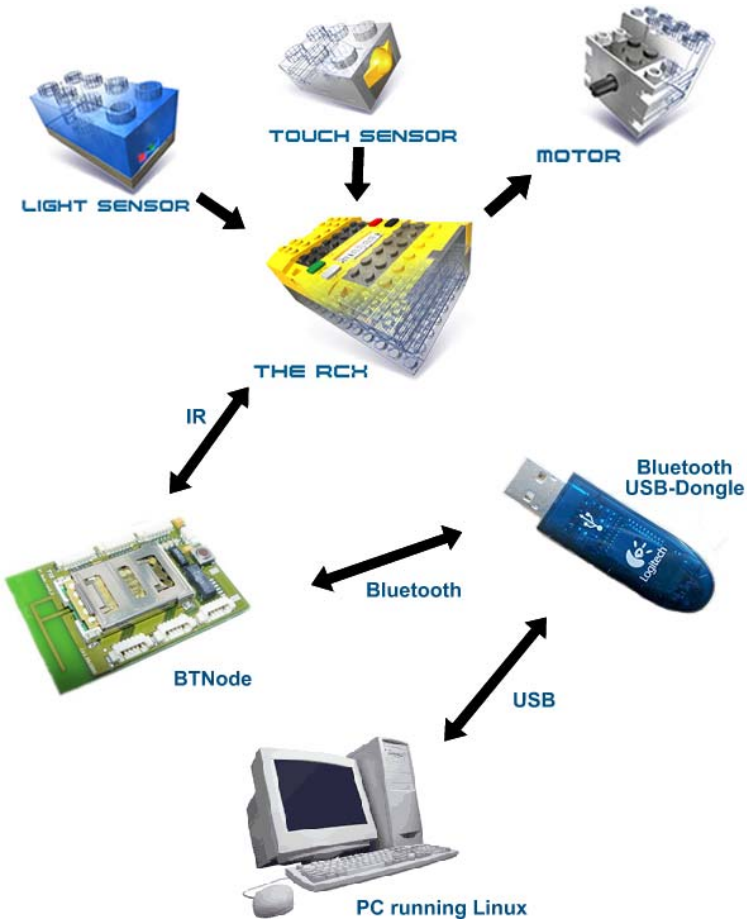


Figure 5: The Communication Path. Displayed are the devices involved in the flow of information and the technologies used for their communication.

For that reasons the Mindstorms Admin Framework handles all RCX communication specific formatting on the level closest to the RCX; in the BTnode controller program. There the RCX packets are parsed and the essential data is extracted and forwarded to the console application using the L2CAP protocol. Messages passing the BTnode in the other direction are likewise encoded according to the RCX packet protocol before they are passed on to the RCX.

3.2.2 Console Application-side Processing

In the console application incoming messages first are separated into two groups: replies to previously sent commands and signals sent by RCX bricks. Replies will be checked for error messages while signals will be handled according to the forwarding policy. Error messages will result in a notification of the user and in the deletion of all messaged scheduled to be sent next to the respective RCX. Replies indicating successful completion of commands will trigger the sending of a message that is scheduled to be sent next and may also result in the notification of the user if the reply indicates the completion of a series of commands which in sequence form one logical user action.

The mechanism for sending commands to an RCX is rather straight forward: the command has to be expressed in RCX byte code and sent via the Bluetooth L2CAP layer to the BTnode handling the RCX. The console application will add a short header to the message to instruct the BTnode to forward the command to the RCX and to indicate how long the expected reply to the message will be.

3.2.3 Example: Inquiring the Battery Power

As an example, let us look at the proceedings following a user's inquiry of the battery power of an RCX brick. The console application first creates the appropriate RCX message. The expected reply to this message is a two-byte message (the current voltage of the batteries). The header of the message will be the message type (forward-to-RCX) and the size of the expected reply data (2 bytes). Because of the Bluetooth L2CAP layer can preserve packet boundaries there is not need for any additional information.

When the BTnode receives the message it first inspects the message type. In the case of this example it translates the message data to the RCX protocol and forwards it to the RCX. The reply of the RCX brick is expected to carry 2 data bytes. Those are in turn forwarded to the console device once the BTnode receives them.

3.3 Interfaces

From the user's perspective, the Mindstorms Admin Framework has two interfaces defined: the user interface of the console application and its module interface. Both are in fact two gateways to one and the same interface but the two gateways underlie different rules of what is allowed and what not.

The user interface is a simple command line interface, as known from many applications throughout the UNIX world. The user types the commands and their parameters and the console application reacts accordingly. All information coming from the RCX or the modules is simply written out to the command line. The user has the permission to use all commands and to steer any RCX and he or she can also set the permissions for the modules. Additionally the user can also use some general commands which affect the console application itself, such as the `broadcast` command which switches the signal broadcasting mechanism on or off.

The module interface works in almost the same way as the user interface. The differences are that some commands, such as the `run` command, are only accepted by the console application if the module has the right to steer the RCX to which the command is to be sent to. Other commands, such as the `exit` command, are always blocked by the console because they also affect other modules and the console session itself. Module interfaces also do not show all information but rather only those related to RCX bricks for which the user has given them permission.

The module interface uses the functionality of UNIX files to allow the attaching of new modules at runtime. For each module two files, one for each direction of communication, must be used because UNIX files do not allow duplex communication. It is worth noting that

UNIX files do not necessarily have to be regular files. Most modules will probably make use of named pipes to connect to the console application.

3.4 Summary

The functionality of the Mindstorms Admin Framework is distributed among three devices. The RCX is the controller of the constructed robot or machine; it hosts the control program which processes the sensor inputs and steers the motor outputs. The BTnode is the connecting link between the RCX and the console application. It is responsible for the registration of the RCX and for translating the messages sent between the RCX and the console. Each one RCX and one BTnode form a BTnode/RCX pair. The console application maintains the connections to BTnode/RCX pairs and hides communication related details from the user and the modules.

The central device of the communication path is the BTnode. It encodes and decodes packets from and to the RCX packet protocol. The connection between a BTnode and the console application uses the L2CAP Bluetooth layer and relies on it to ensure a fault free communication. The console application decides on the next message to send based on the incoming reply packets. Signal packets are handled according to the signalling policy. The user is notified if commands could not be executed.

The console has two, very similar, interfaces. The user interface has permission to use all commands of the console on all RCX bricks and all modules. The module interface has only permissions to see and steer those RCX bricks which the user has allowed it to. The module interface uses UNIX files, two for each module.

4 Implementation

The next two sections discuss the important issues of the implementation. For each of the two devices that were programmed a section explains some of the algorithms and data structures that were used.

4.1 BTnode Controller

The implementation of the BTnode control program consists of two areas. The first area handles the establishment and maintenance of a connection to the console application while the second area handles the actual communications between the RCX on one side and the console application on the other.

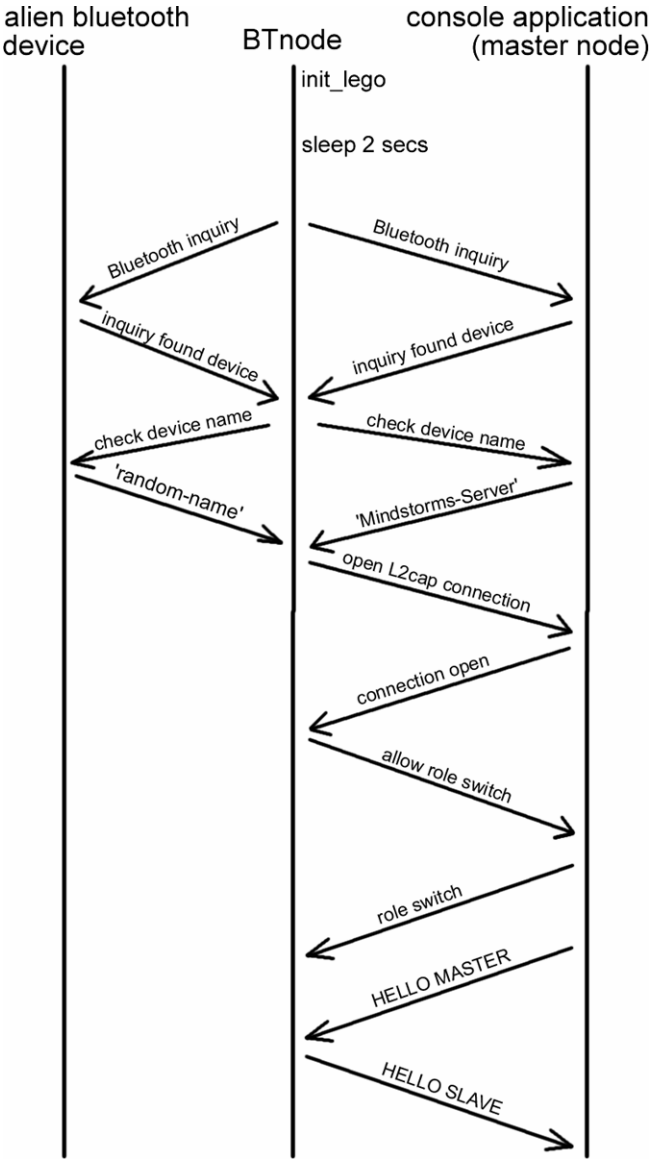


Figure 6: A schema of the connection initialisation mechanism.

4.1.1 Connection Management

Upon start-up the BTnode will automatically go into the connection initialisation mode in which it will try to find an appropriate Bluetooth device to act as its master and to establish a stable connection to it.

The BTnode (logically acting as the client) will try to find the Bluetooth device of the console application (logically acting as the server) by initiating a Bluetooth inquiry. Then it will try to connect to PSM 185 of the device named ‘Mindstorms-Server’ (see figure 6), if such a device exists. If the establishment of a connection succeeds the BTnode (now in Bluetooth master mode) will signal it is ready to make a role switch to slave mode and the server will automatically try to go to master mode. A simple handshake mechanism, triggered by the server, will then ensure that both peers really belong together.

Making a role switch is important because the Bluetooth device initiating the connection is the master by default and the device connected to is the slave by default. However Bluetooth slave devices seem to ignore inquiries so that any further searches by other BTnodes after the first connection has been established would not find the device representing the server.

The BTnode will close the connection and restart the connection initialisation mechanism if the procedure fails at some point. It will

inquire for Bluetooth devices several times, each time for a longer interval, but eventually it will stop inquiring after about a minute if no master node was found and issue a buzz beep on the RCX (if one is connected) to indicate a failure.

If a master has been found the BTnode will issue an upwards tune beep on the RCX (if one is connected) to indicate success and will go into a passive state, in which it only acts in response to either RCX side signals or to incoming Bluetooth packets.

If the connection to the master is closed during passive mode, the BTnode will ensure a clean shutdown of the respective channel, issue a downwards tune beep on the RCX (if one is connected) to indicate loss of connection to the master node and return to the connection initialisation mode.

4.1.2 BTnode-to-RCX Communication

The original Proof-of-Concept, on which the implementation of the BTnode controller is based, uses the SUART (Universal Asynchronous Receiver-Transmitter with software controller) to communicate with the infra-red interface of the RCX. A short check indicated that this interface misreads incoming bits in a much too high rate of over 10% bit errors. This was assumed to occur because the bit scanning of the SUART interface is done in software and is therefore relatively slow in comparison to the bit transmission rate. The assumption proved true: when instead the UART interface with its hardware scanning was used, the bit error rate dropped to almost 0%. The change of the infra red communication from SUART to UART already represents a big improvement in terms of reliability compared to the Proof-of-Concept.

A second major change to the RCX communication was the complete rework of the RCX input parser and changes to the RCX-encoder. In the Proof-of-Concept the parser was correct, but it was not capable of handling anything else than signal messages. The redesign ensured that the BTnode can forward any message type from the RCX to the console application.

Sending messages to the RCX is rather trivial: a constant preamble of three bytes is sent, then the data bytes with their negation bytes immediately following. The BTnode also remembers the first byte of the last sent message and compares this value with the first byte of the next message. If the values are the same - i.e. the message type of the two messages is the same - the third bit (the 0x08 bit) will be flipped to ensure that the RCX does not mistake the message for an echo of the last sent message.

4.1.3 RCX-to-BTnode Communication

The parser for the input from the RCX brick on the other hand is a little bit more complex because neither SUART nor UART can identify package borders of RCX messages. The parser has to be implemented as some sort of an abstract state machine (ASM) so that it can retain its state after parsing one group of incoming bytes and continue in the same state when the next group of bytes has been detected.

As implemented during this project the parser is not an ASM in its strict definition because it has multiple states at the same time: one state separating message header parsing from message body parsing, one separating parsing of echo from parsing of replies and parsing of RCX triggered messages and finally one binary state separating parsing of data bytes from parsing of negation bytes.

The implemented parser is capable of handling any message, as long as it knows how long the message is supposed to be. Because the BTnode should not be aware of the meaning of the message it forwards it can also not know how long the reply to the message will be. For this reason the messages incoming from the console that should be forwarded to the RCX always indicate how long the reply message will be. This information is passed as the second byte of the message header.

The parser has the option to ignore incoming messages or to forward them to the server if one is known. By default it will always forward messages. But the server can also request that the reply to a message will be ignored by setting second byte in the header to zero.

Whatever the current policy is, the parser always has to be aware of the current state of the communications because it has to be able to detect the start of the next message even if it ignored the last one. This is however rather trivial because the header sequence is designed to be unique and the parser thus can always reset its state to expect an incoming message header and ignore all bytes not complying to the expected sequence. Thus it can be guaranteed that after scanning of a header sequence the parsers states are always synchronized with the RCX.

4.1.4 Console Communication Management

Once the communication channel to a master device has been established, everything else about this aspect is very simple. There is no need for package management since the Bluetooth protocol is capable of maintaining package borders. The only information needed besides the package data itself is the type of the message (forward-to-RCX, ping or handshake message) encoded in the first byte and the length of the reply expected from the RCX encoded in the second byte.

Forward-to-RCX messages will be forwarded to the RCX essentially unmodified. The length of the expected reply can be indicated in the second byte. If this value is zero the parser will ignore the reply. This functionality has been explained in the last subsection.

Ping messages are sent by the server to verify that the BTnode is still responding. The BTnode always replies with a Pong message. Note that the Ping can be initiated by the either the user (by calling the command `ping`) or by the server. The latter occurs when the server suspects the connection to the BTnode to have died because it detects that the connection has been silent for some time.

Handshake messages are used during the connection initialisation phase. This functionality has been discussed in the subsection on Connection Management.

4.2 Console Application

The console application is the most crucial part in the Mindstorms Admin Framework. It has to coordinate the communication to and between the BTnode/RCX pairs acting as clients, it has to break down complex commands, issued by the user or by modules, into sequences of messages. The command to run a program, for example, is translated into three commands: stop-current-program, select-program, and run-selected-program.

The console application then has to schedule the messages that are waiting to be sent to clients and it has to manage the connected modules and their individual permissions.

4.2.1 Client Management

The first thing that the console application will do is use the socket interface of the BlueZ Bluetooth library to open a L2CAP PSM for listening. It will then enter the main loop where it will make a blocking call to the `select()` function, waiting for new clients to connect or for the user to enter commands on the standard input stream.

When a new client connects, the application will register it by allocating a wrapper data structure to handle the client. This data structure will be marked unready until the handshake message is received. Next the handshake is initiated by send the `LEGO_MASTER` message to the BTnode. The console application then returns to the main loop.

In the main loop, the clients will be checked every few seconds to verify that the connections are still open. If no messages have been received from the client for some time then a a number of ping messages will be sent. If the client still does not reply the application

will assume that the connection broke down and close it. The user and all modules which are allowed to see information regarding the client will receive a notification.

4.2.2 Command Handling

When a module or the user issues a command in form of a line of text then the text will first be parsed. The first word in the text is always the command and the following items are parameters. The first parameter has a special significance since for many commands it indicates the client or the module on which the command should be applied. After the parsing completed successfully, the application will check if the command is valid, i.e. if it is implemented. If the issuer is a module, then the next check will verify if the command is valid for modules or only for the user interface.

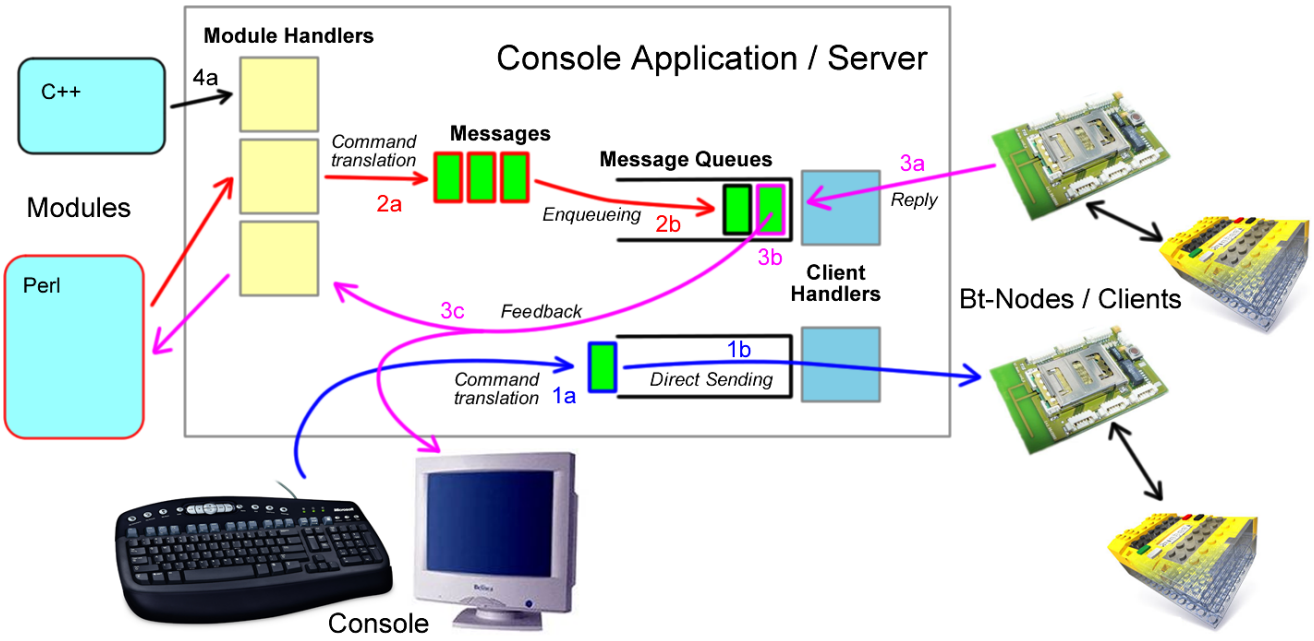


Figure 7: Overview of the implementation of the console application. Internal data structures are labelled in bold font, mechanisms in italics. The arrows indicate flow of information. The figure shows four actions. Blue: The user initiates a simple command which is translated into a single message (1a) and sent directly to the client (1b). Red: A module issues a complex command which is translated into a sequence of messages (2a). The messages then are added to the message queue (2b). Violet: A reply from an RCX (3a) is handled by the first message in the queue (3b) and a feedback message is relayed to the user and a module (3c). Black: A module tries to issue a command for which it has no permission (4a). No messages are generated.

After it has been verified that the command is valid for the issuer the application will check if the command acts on a BTnode/RCX pair or on modules and if it does it will inspect the first parameter to check if the parameter specifies a connected client or module or if it is the code word all which indicates that all valid clients or modules are to be targeted in which case a copies of the command will be issued; one for each valid node or module.

Once it is sure that the command is an allowed command for the issuer and that the target of the command exist the actual command will be translated into one or several messages and the messages passed on to the message scheduler. This mechanism varies a lot from one command to another. The `beep` command for example is translated in a straight forward manner into one message while the translation of the `upload` command includes the loading

of a byte code file, its parsing and packing into ten or more messages, depending on the complexity of the program.

A sequence of messages will often be constructed in a manner in which one message evaluates the reply of the proceeding message and terminates the sequence with a notification if an error occurred. Because the console application runs in only one process there is no possibility that two message sequences mix or overlap so that a message can always assume that the preceding message is also the preceding message of the sequence if the sequence indeed has a preceding message.

4.2.3 Message Scheduling

Message scheduling is implemented with one message queue per client. This is a FIFO (first in first out) data structure that ensures that the messages will be processed in the same order as they have been added. The command execution has the possibility to either add a message in the queue regardless of its state or to try to send a message directly. In the latter case the scheduler will actually send the message only if the queue is empty. If there are already messages waiting to be sent then the scheduler will add the message at the end of the queue.

When replies to messages arrive at the console the scheduler will give the first message in the queue the opportunity to inspect the reply and to report an error if the reply indicates that something went wrong. If so, then all messages waiting in the queue will be removed to guarantee that no fragments of message sequences remain in the queue after one of the messages of the sequence failed. The first message of the queue can then send data (usually itself) to the client and it can display information which will automatically be shown to the user and to all modules that are allowed to see information regarding the client to which the queue is associated.

In case that the message does not send data nor report an error the scheduler will assume that the message is just a “reply-consumer”, marking the end of a message sequence. The scheduler will therefore send the message next in the queue, if there is one.

4.2.4 Module Management

Modules each have a wrapper structure very similar to the structures used for clients. The most important differences are that there are two types of module connections, input and output and that the application must store information regarding the permissions of the modules.

The type of the connection is set when the internal data structure is created with the first permission defining command. Subsequent permission setting commands on the same connection have to be of the same type. Unlike the management of the clients there is no check for reaction of connections to the modules since this is not possible for unidirectional communication. The console application relies on the stability of the modules.

Interactive modules that need to receive information as well as steer RCX bricks will have to use two files, to communicate with the console application; one for input and one for output. It is also strongly advised to use pipes instead of regular files since those are better suited for the purpose.

4.3 Summary

The implementation of the BTnode comprises three main areas. The connection management searches and establishes a connection to the master Bluetooth device. A role switch is needed because the master device would otherwise not be visible to further searching BTnodes. Communication with the RCX is established using a simple infra red transmitter device. The RCX packet parser is responsible for the extraction of the essential

data of an RCX message. Before being forwarded to the RCX brick the messages coming from the console application have to be encoded according to the RCX packet protocol. The communication with the console application uses the L2CAP Bluetooth layer because of its reliability which greatly simplifies the work of the BTnode.

The console application manages the connections to BTnode/RCX pairs (clients) and to modules and it translates commands into sequences of messages which are scheduled to guarantee that no sequences overlap. Message replies are checked for indications of errors. Modules interact with the console application using one-way connections in form of UNIX files.

5 Results

This chapter discusses the results achieved by showing how a short session could look like. The second section lists some results related to the stability and reliability of the Mindstorms Admin Framework. Issues that have not been handled follow up in the third section.

5.1 Example of Use

To demonstrate the use of the Mindstorms Admin Framework an example application is given in the following.

First the console application is started:

```
[wehrlil@pc-3631 mindstorms_admin_framework]$ ./console
Mindstorms Admin Console at Bluetooth device [fe:d6] entering main
loop..
[fe:d6] >
```

After connecting two BTnodes to RCX bricks the console application detects the clients. The first is assigned to the number 4, the second to the number 5. To allow the user to identify the two RCX bricks the console commands them to emit a short tune. Thus the user knows that the RCX beeping first is the one with the number 4 and the second the number 5.

```
[fe:d6] > (4) > NEW NODE [4d:14]
[fe:d6] > (4) > NEW NODE BEEPING
[fe:d6] > (5) > NEW NODE [4d:1f]
[fe:d6] > (5) > NEW NODE BEEPING
```

To check if the BTnodes really work properly, the user first sends pings to all connected clients. The command `ping all` does just that:

```
[fe:d6] > ping all
[fe:d6] > (4) > PONG
[fe:d6] > (5) > PONG
```

Both devices answer correctly with a PONG message. The numbers in braces indicate to which RCX the following information relates. Next, the user wants to upload a short test program to both RCX bricks. The byte code of the program is stored in the file test.rcx. The program should reside in the slot number 3 of the RCX.

```
[fe:d6] > upload all test.rcx 3
[fe:d6] > (5) > UPLOAD 3
[fe:d6] > (4) > UPLOAD 3
```

The user wants to execute the uploaded programs. The command `run`, followed by the RCX bricks and the program slot number lets the RCX stop the currently running programs, change to the program slot number, and run the program.

```
[fe:d6] > run all 3
[fe:d6] > (5) > STOP ALL TASKS
[fe:d6] > (4) > STOP ALL TASKS
```

```
[fe:d6] > (5) > SET PROGRAM 3
[fe:d6] > (4) > SET PROGRAM 3
[fe:d6] > (5) > RUNNING PROGRAM 3
[fe:d6] > (4) > RUNNING PROGRAM 3
```

The program test.rcx waits for incoming signals and reacts to them by playing tunes and by sending back a signal. The number of the sent signal is the one received less one. Because the automatic broadcasting of signals is on by default, the signals received will be routed to the other RCX brick. A ping pong effect results in which the numbers of the signals decrease after each step. The user starts the mechanism by sending the signal 5 to the RCX with the number 5.

```
[fe:d6] > signal 5 5
[fe:d6] > (5) > SIGNAL 4
[fe:d6] > (5) > SIGNAL 4 REROUTED TO NODE 4
[fe:d6] > (4) > SIGNAL 3
[fe:d6] > (4) > SIGNAL 3 REROUTED TO NODE 5
[fe:d6] > (5) > SIGNAL 2
[fe:d6] > (5) > SIGNAL 2 REROUTED TO NODE 4
[fe:d6] > (4) > SIGNAL 1
[fe:d6] > (4) > SIGNAL 1 REROUTED TO NODE 5
```

By sending the signal 5 to both RCX bricks, the user can make the bricks “play ping pong with two balls”:

```
[fe:d6] > signal all 4
[fe:d6] > (5) > SIGNAL 3
[fe:d6] > (5) > SIGNAL 3 REROUTED TO NODE 4
[fe:d6] > (4) > SIGNAL 3
[fe:d6] > (4) > SIGNAL 3 REROUTED TO NODE 5
[fe:d6] > (5) > SIGNAL 2
[fe:d6] > (5) > SIGNAL 2 REROUTED TO NODE 4
[fe:d6] > (4) > SIGNAL 2
[fe:d6] > (4) > SIGNAL 2 REROUTED TO NODE 5
[fe:d6] > (5) > SIGNAL 1
[fe:d6] > (5) > SIGNAL 1 REROUTED TO NODE 4
[fe:d6] > (4) > SIGNAL 1
[fe:d6] > (4) > SIGNAL 1 REROUTED TO NODE 5
```

Finally the user stops the running program. He or she also wants to know the power of the batteries.

```
[fe:d6] > stop all
[fe:d6] > (5) > STOP
[fe:d6] > (4) > STOP
[fe:d6] > battery all
[fe:d6] > (5) > BATTERY: 6795mV
[fe:d6] > (4) > BATTERY: 6682mV
```

Because the battery power is low, the user decides to end the session. All connections are automatically closed before the console application terminates.

```
[fe:d6] > exit
[fe:d6] > (5) > CLOSE (USER)
[fe:d6] > (4) > CLOSE (USER)
[fe:d6] > bye bye!
[wehrlil@pc-3631 mindstorms_admin_framework]$
```

5.2 Stability Issues

Improvement of stability was the primary objective for the Mindstorms Admin Framework project. Particularly the BTnode controller program and the use of the BTnode IR device in the Proof-of-Concept were to be improved. But the console application too should run reliable even if the BTnodes are unreliable. The following statements base on experiences made with the Mindstorms Admin Framework during the project work itself.

- The connection between the BTnode and the RCX has been greatly improved by the move of the IR device from SUART to UART. Where the communication was unreliable and faulty in the Proof-of-Concept the final set-up of the Mindstorms Admin Framework is reliable to the extent that not a single transmission error was observed after the modification.
- In the Proof-of-Concept the BTnode has the annoying habit of regularly loosing connections in an unpredictable manner. This behaviour disappeared completely after some errors in the controller were corrected.
- The console application runs absolutely reliable. It has been tested to maintain connections to multiple BTnode/RCX pairs over long time periods. Wrong user inputs are caught and handled correctly.
- Extensive usage of the connections does not cause malfunctions.
- The `upload` command sometimes causes the BTnode to crash. The same error appears also in other BTnode applications. It is most probably caused by a bug in the BTnode API.

The Mindstorms Admin Framework allows the user to forget about the details of the communication between the RCX bricks. Instead users can concentrate on the more important issues of their work. The problem with the program upload is acceptable since uploading is not a task that is usually performed during run-time.

5.3 Open Issues

During the project duration some tasks remained unfinished because of lack of time or because they where not deemed as important as others. The most important of those issues are the following:

- A graphical display of the RCX bricks and the messages they send to each other was not implemented because the console proved to be very clear and sufficient in its presentation of information.
- Not all RCX commands can be used by the console application. While clearly not all commands make sense in this context there are some which could be useful for some exotic applications.
- Some problems with the BTnode could not be solved. It seems that a bug in API is causing the BTnode to crash on some occasions.

5.4 Summary

In this chapter the reader first was shown how a simple session on the console application, using two connected RCX/BTnode pairs, could look like. This sample included the upload of a program, its execution and termination as well as communication with the RCX program using signals. The broadcasting functionality was also demonstrated.

The stability issues listed clearly testify to the reliability of the Mindstorms Admin Framework. It allows users to focus on the main aspects of their work instead of fussing with the communications.

The section on open issues shows up how the framework could be extended in the future. While there are no pressing issues, there are some features which could still be added, notably the support of some RCX commands not yet implemented.

6 Conclusions

The following chapter closes this document by listing some conclusions of the project work. The first section lists the conclusions related to the project as a whole, while the second section focuses on the evaluation of the approach.

6.1 Overall Conclusions

The Mindstorms Admin Framework is great tool for anyone working with multiple RCX bricks. The framework allows for completely new application areas to be explored with the LEGO Mindstorms Robotics Invention System. In many tasks, where before the Mindstorms robots seemed inappropriate or insufficient, the Mindstorms Admin Framework can be used to produce elegant solutions with ease.

The constraints of communication in the original LEGO Mindstorms system are overcome; the RCX bricks do no longer need to be arranged in a sight contact. Instead they can be arranged in any position or they can move freely, while still maintaining reliable communications.

The User does not have to take care of technical details. Instead, he can concentrate on the construction and controlling of the robots. The console interface allows the user to take full control over all RCX bricks connected and to monitor the communicated signals. He can inquire, amongst others, the internal state of the RCX, the voltage of its batteries or the value of the attached sensors. This is even possible while a program is running on the RCX.

Defining a central coordination for the RCX bricks is easy; any programming language can be used to implement a controller program, which is then connected to the module interface. A strict permission system ensures that no module can see or access robots without explicit permission from the user.

6.2 Approach

When looking back on the design decisions taken at the start of the project, it gets clear that those where appropriate and in the end resulted in a very good product. The decision to leave the LEGO Mindstorms Robotic Invention System as intact as possible simplifies the work with the final product tremendously; the user can apply well-known and ripe technologies instead of having to learn a completely new system.

The use of a Client/Server architecture and of the BTnode as intermediary between RCX bricks and the console application simplified the whole work because of the clear distribution of tasks amongst the different components. The resulting system is easy to understand and its dynamics can be interpreted in a straight forward manner.

Using NQC programs has many advantages to: the byte code files are simple and easy to parse. The programs are powerful in their many ways to solve problems while at the same time being compatible to the original RCX firmware. The programming language NQC is easy to learn because of its syntactical proximity to C, one of the best known programming languages.

While a different approach could have been taken to give the user even more control over the RCX (for example by using a different firmware/OS), it is very doubtable if the results would have surpassed the Mindstorms Admin Framework in terms of user friendliness. User

friendliness, however, was a key requirement of the project. Thus the approach taken for the Mindstorms Admin Framework can be said to be fully justified.

6.3 Outlook

Although the Mindstorms Admin Framework is a complete system with powerful features, there still are some features that could be extended or added. Future work could comprise some of the following tasks:

- Extending the set of commands of the console application to include some yet unsupported RCX commands for inquiring the state of an RCX brick or of the attached sensors.
- Allowing the upload of firmware updates. This seems a rather simple task since the upload of firmware works very similar to the upload of programs.
- Modifying the BTnode controller program to allow a peer-to-peer connection between BTnodes. The design of robot communication in a peer-to-peer network could be of some interest.

Zurich, February 23, 2004

Lior Wehrli
wehrlil@student.ethz.ch

7 References

[1] LEGO Corporation, **LEGO Robotics Invention System**. http://mindstorms.lego.com

[2] ETH Zürich, Institut für Technische Informatik und Kommunikationsnetze, **Studentenprojekte mit LEGO Mindstorms Robotic Invention System**. http://www.tik.ee.ethz.ch/mindstorms/

[3] The BTnode Project, **BTnodes – A Distributed Environment for Prototyping Ad Hoc Networks**. http://btnode.ethz.ch

[4] **BlueZ**, Official Linux Bluetooth protocol stack. http://www.bluez.org/

[5] David Baum, **Not Quite C**. http://bricxcc.sourceforge.net/nqc/

[6] Kekoa Proudfoot, **RCX internals**. http://graphics.stanford.edu/~kekoa/rcx

Appendix A: HOWTOs

Installation & Configuration HOWTO

This HOWTO describes how to install and configure the Mindstorms Admin Framework.

Requirements

To install and use the Mindstorms Admin Framework you need a distribution of the Mindstorms Admin Framework and a working installation of the BlueZ Bluetooth protocol stack. This is the standard Bluetooth protocol stack for Linux since as of kernel 2.4, so if you have a recent Linux distribution, you probably already have this installed. This software was written with bluez-libs 2.3 and bluez-utils 2.2. If you do not have an installation of BlueZ or if you want to get the newest versions, you may download the current distribution from http://www.bluez.org.

Configuring the Bluetooth device

Make sure a Bluetooth device is connected to the computer, for example over USB if you use a USB dongle. Then run `hciconfig` to get a listing of the connected devices. Notice: Depending on the configuration of your system, you may have to be root to use that command. If you have to be root but you do not know how do get root, ask your administrator for help. The console should now display some general information on the connected Bluetooth devices. This should look more or less like that:

```
[root@pc-3631]# hciconfig
hci0:   Type: USB
        BD Address: 00:60:57:18:FE:D6 ACL MTU: 192:8  SCO MTU: 64:8
        UP RUNNING PSCAN ISCAN
        RX bytes:99 acl:0 sco:0 events:13 errors:0
        TX bytes:296 acl:0 sco:0 commands:12 errors:0
```

If instead you get something like that

```
[root@pc-3631]# hciconfig
hci0:   Type: USB
        BD Address: 00:00:00:00:00:00 ACL MTU: 0:0  SCO MTU: 0:0
        DOWN
        RX bytes:0 acl:0 sco:0 events:0 errors:0
        TX bytes:0 acl:0 sco:0 commands:0 errors:0
```

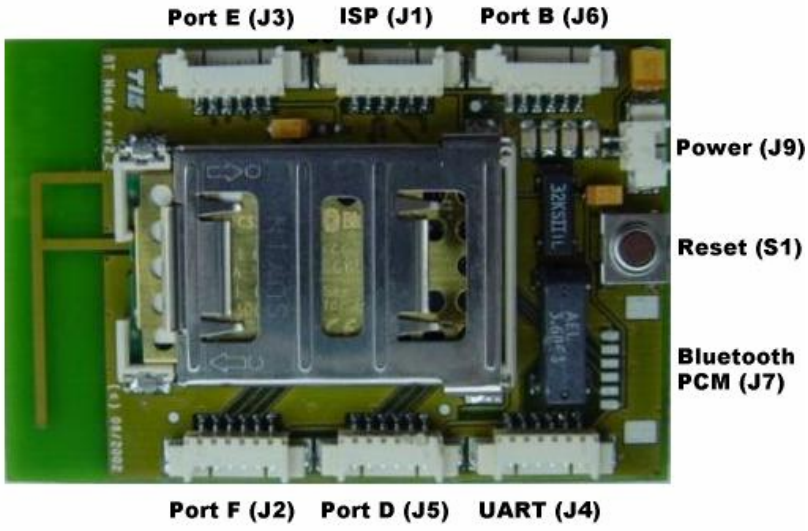
.. then the Bluetooth device was not found. It often helps to remove the device and plug it in again. If that still did not help, you can run `hcid` to let the kernel search for devices. If your system still can not find the device you will have to refer to the BlueZ documentation on http://www.bluez.org/documentation.html for help.

You now have to configure the Bluetooth device according to the needs of the Mindstorms Admin Console. Type the following commands:

```
hciconfig hci0 lm MASTER,ACCEPT
hciconfig hci0 name 'Mindstorms-Server'
```

Setting up the BTnode

You should now connect an IR-device to each BTnode. Use the UART (J4) connector. The IR transceiver has only one connector, so there should be no problem there. Fix the IR-transceiver directly on the IR-interface of the RCX brick. You may want to use a LEGO construction for that or a rubber.



The power supply for the BTnode must be connected to J9. If you use the batteries of the RCX, you should disconnect the BTnode when you do not work with it. Unlike the RCX, the BTnode will not shut itself off by itself after some time. The batteries of the RCX will be drained quickly, if you leave the BTnode connected to them longer than necessary.

The blinking of the innermost LED indicates that the BTnode is running correctly. If any other LEDs blink the BTnode has crashed and you should press the reset button.

Starting the Console Application

Now you are ready to run the console application. Unpack the Mindstorms Admin Framework distribution to a directory of your choice and enter that directory. Then type

```
console
```

If you have not connected the BTnodes to their power supply, you can do so now. After a short wile, the BTnodes should have found the console application and you are ready to start your work with the Mindstorms Admin Framework.

Program Control HOWTO

This HOWTO describes how to upload, run and communicate with programs on RCX bricks.

The first thing you have to do is to run the console application in the directory in to which you have installed the Mindstorms Admin Framework.

```
[wehrlil@pc-3631 mindstorms_admin_framework]$ ./console
Mindstorms Admin Console at Bluetooth device [fe:d6] entering main
loop..
[fe:d6] >
```

If the console application does not display the same line as above or the address displayed is [00:00], then you should close the console application (type `exit`) and follow the instructions of the Installation and Configuration HOWTO.

Now make sure that the BTnodes are connected to their power supplies and that the RCX bricks are turned on. The BTnodes should find the Bluetooth device of the console application within a minute. If they do not , you have to press the reset button of the BTnodes.

```
[fe:d6] > (4) > NEW NODE [4d:14]
[fe:d6] > (5) > NEW NODE [4d:1f]
[fe:d6] > (4) > NEW NODE BEEPING
[fe:d6] > (5) > NEW NODE BEEPING
```

The BTnodes have now signed themselves in to the console application. To check if the nodes are still responding you can use the `ping all` command.

```
[fe:d6] > ping all
[fe:d6] > (4) > PONG
[fe:d6] > (5) > PONG
```

If you do not know the address assigned to the RCX bricks you can use the `beep` command to let a RCX play a short tune. The first argument of the `beep` command is the number the node, the second is the number of the tune to be played.

```
[fe:d6] > beep 4 1
[fe:d6] > beep 5 2
```

To get information about the commands at your disposal, use the `help` command. To get an explanation on a specific command use `help <com>`, where `<com>` is the command.

The next thing to do (usually) is to upload a program to the RCX bricks. The file test.rcx contains a simple program that can be used to test uploading and signalling.

```
[fe:d6] > upload all test.rcx 3
[fe:d6] > (5) > UPLOAD 3
[fe:d6] > (4) > UPLOAD 3
```

The `upload` command takes three arguments: the RCX bricks' number, the program file name and the program slot, to which the program will be uploaded to. Because uploading is a very bandwidth intensive operation, problems may occur here. If so, do not be discouraged; try again.

Once the upload has completed successfully, you can run the program as such:

```
[fe:d6] > run all 3
[fe:d6] > (5) > STOP ALL TASKS
[fe:d6] > (4) > STOP ALL TASKS
[fe:d6] > (5) > SET PROGRAM 3
[fe:d6] > (4) > SET PROGRAM 3
[fe:d6] > (5) > RUNNING PROGRAM 3
[fe:d6] > (4) > RUNNING PROGRAM 3
```

The `run` command is executed in three stages, each of which displays a feedback message on the console. If the console looks like above, then the execution of the program was started correctly.

The program in test.rcx is very simple: it waits for incoming signals between 1 and 5, plays a tune melody and sends a signal out. The number of the sent signal is the number of the received signal minus one. For example, if a signal of number 4 is received, then a signal of number 3 will be emitted.

In the console application, automatic signal broadcasting is turned on by default. You can use the `broadcast` command to switch broadcasting on and off, but for the next step it has to be turned on.

Use the `signal` command to send the signal 5 to one of the RCX bricks.

```
[fe:d6] > signal 5 5
```

```
[fe:d6] > (5) > SIGNAL 4
[fe:d6] > (5) > SIGNAL 4 REROUTED TO NODE 4
[fe:d6] > (4) > SIGNAL 3
[fe:d6] > (4) > SIGNAL 3 REROUTED TO NODE 5
[fe:d6] > (5) > SIGNAL 2
[fe:d6] > (5) > SIGNAL 2 REROUTED TO NODE 4
[fe:d6] > (4) > SIGNAL 1
[fe:d6] > (4) > SIGNAL 1 REROUTED TO NODE 5
```

As you can see, the signal triggered a ping-pong effect: signals are sent from one RCX to the other until the stepwise decrease of the signal number arrives at signal number 1. The broadcasting mechanism reroutes each incoming signal from one RCX to the other RCX like a ping-pong ball going from one player to the other.

You can also do the same “with two ping-pong balls” by sending signals to both RCX bricks at the same time.

```
[fe:d6] > signal all 5
[fe:d6] > (5) > SIGNAL 4
[fe:d6] > (5) > SIGNAL 4 REROUTED TO NODE 4
[fe:d6] > (4) > SIGNAL 4
[fe:d6] > (4) > SIGNAL 4 REROUTED TO NODE 5
[fe:d6] > (5) > SIGNAL 3
[fe:d6] > (5) > SIGNAL 3 REROUTED TO NODE 4
[fe:d6] > (4) > SIGNAL 3
[fe:d6] > (4) > SIGNAL 3 REROUTED TO NODE 5
[fe:d6] > (5) > SIGNAL 2
[fe:d6] > (5) > SIGNAL 2 REROUTED TO NODE 4
[fe:d6] > (4) > SIGNAL 2
[fe:d6] > (4) > SIGNAL 2 REROUTED TO NODE 5
[fe:d6] > (5) > SIGNAL 1
[fe:d6] > (5) > SIGNAL 1 REROUTED TO NODE 4
[fe:d6] > (4) > SIGNAL 1
[fe:d6] > (4) > SIGNAL 1 REROUTED TO NODE 5
```

You can play around with the RCX and the control program. When you had enough, use the `stop` command to stop the execution of the program.

```
[fe:d6] > stop all
[fe:d6] > (5) > STOP
[fe:d6] > (4) > STOP
```

Do not forget to use the `battery` command to check on the battery power of the RCX bricks from time to time. A fully charged pack of batteries has a voltage to 9V and the minimal voltage required for the RCX to run is approximately 6500 mV.

```
[fe:d6] > battery all
[fe:d6] > (5) > BATTERY: 6795mV
[fe:d6] > (4) > BATTERY: 7282mV
```

Finally, to close the console and all connections, use the `exit` or `quit`.

```
[fe:d6] > exit
[fe:d6] > (5) > CLOSE (USER)
[fe:d6] > (4) > CLOSE (USER)
[fe:d6] > bye bye!
[wehrlil@pc-3631 mindstorms_admin_framework]$
```

Module Connection HOWTO

This HOWTO describes how use the module interface of the Mindstorms Admin Framework. The file smpl_coord contains a script, written in Perl, that, when executed, establishes two named pipes. The pipe coord-in is for console-to-module communication, the pipe coord-out for module-to-console communication.

After running smpl_coord, the connections from the console to the module are established with the commands `route` and `allow`. Both commands take the number assigned to an RCX brick as first argument and the name of the file or pipe as a second. Nodes thus associated with files or pipes can be controlled from outside modules. For more information, type the command `help route` and `help allow`.

Execute the module in a separate terminal before connecting it to the console! Otherwise, the console will block or even terminate, because it tries to write to a pipe without reader. The module smpl_coord on the other hand is passive, so it will block until messages are sent to it by the console. Never connect the console to pipes that do not have a reader or a writer.

To close the module manually you have to press Ctrl C, because the script does not check its standard input. It is strongly advised to use the smpl_coord module as a basis for development of own coordination modules.

Start the console application, turn one of the RCX bricks on and connect its BTnode to its power supply. It is assumed, that the RCX brick has the program test.rcx uploaded to slot 3. If that is not true, follow the instructions of the Program Control HOWTO.

```
[wehrlil@pc-3631 mindstorms_admin_framework]$ ./console
Mindstorms Admin Console at Bluetooth device [fe:d6] entering main
loop..
[fe:d6] > (4) > NEW NODE [4d:14]
[fe:d6] > (4) > NODE BEEPING
```

Next route all incoming information from the RCX device to the pipe coord-in with the `route` command and allow external modules using the pipe coord-out to control the RCX by calling the `allow` command.

```
[fe:d6] > route 4 coord-in
[fe:d6] > allow 4 coord-out
```

The script smpl_coord is a example of how a central coordination module could be implemented with the Perl script language. Since this module takes care of the broadcasting of signals, you have to turn off the automatic signal broadcasting mechanism using the command `broadcast`.

```
[fe:d6] > broadcast off
[fe:d6] > BROADCAST OFF
```

Then run the program in slot 3 (assumed to be test.rcx) and begin the “ping-pong”-process by sending a signal to the RCX.

```
[fe:d6] > run 4 3
[fe:d6] > (4) > STOP ALL TASKS
[fe:d6] > (4) > SET PROGRAM 3
[fe:d6] > (4) > RUNNING PROGRAM 3
[fe:d6] > signal 4 5
[fe:d6] > (4) > SIGNAL 4
[fe:d6] > (4) > SIGNAL 2
```

As you can see, the RCX beeps more then once, indicating that it got more than just the first signal. That is because the module smpl_coord received the signals sent by the RCX brick and responded to them by sending back a signal of decreased number. Have a look into the file smpl_coord to see how this was done.

You can now stop the running program and close the console as described before. This will also terminate the module, since it tries to read from a pipe without writer.

```
[fe:d6] > stop 4
[fe:d6] > (4) > STOP
[fe:d6] > off 4
[fe:d6] > exit
[fe:d6] > (4) > CLOSE (USER)
[fe:d6] > CLOSING MODULE coord-out
[fe:d6] > CLOSING MODULE coord-in
[fe:d6] > bye bye!
[wehrlil@pc-3631 mindstorms_admin_framework]$
```

The terminal in which you executed the module script should approximately look like that:

```
[wehrlil@pc-3631 mindstorms_admin_framework]$ ./smpl_coord
Node 4 has adress [4d:14]
Node 4 sent command STOP with params ALL TASKS
Node 4 sent command SET with params PROGRAM 3
Node 4 sent command RUNNING with params PROGRAM 3
Node 4 sent command STOP
Node 4 sent command CLOSE with params (USER)
[wehrlil@pc-3631 mindstorms_admin_framework]$
```

Appendix B: Console Command Reference

I. General commands

help

Usage:

```
help
help <com>
```

The `help` command displays information about a command `<com>` or lists all known commands on the console.

list

Usage:

```
list
```

The `list` command lists all connected nodes with their addresses and all connected modules with their associated nodes to the console.

ping

Usage:

```
ping <node>
ping all
```

The `ping` command sends a ping message to a connected BTnode `<node>` or to all connected BTnodes, if the argument `all` is used.

timeout

Usage:

```
timeout
timeout <secs>
```

The `timeout` command sets the timeout value for the message queues to `<secs>` seconds, when used with a parameter, or displays the current timeout value. The default value is 10 seconds. The timeout of the message queues is the amount of time, that the console application waits for a reply to a sent message before declaring the message as failed.

exit / quit

Usage:

```
exit
quit
```

The `exit / quit` command closes all connections to modules and nodes and terminates the console application. `exit` and `quit` are synonyms.

close

Usage:

```
close <node>
close <module>
close all
```

The `close` command closes the connection to a node `<node>` or to a module `<module>` or to all connected nodes and modules, if the argument `all` is used. Closing nodes does not change the permissions of modules associated with the node.

II. RCX Program Handling

upload

Usage:

```
upload <node> <file> <slot>
upload all <node> <file> <slot>
```

The `upload` command uploads a byte code file `<file>` to a program slot `<slot>` of the node `<node>` or of all connected nodes, if the argument `all` is used. Uploading is a complex task and may require more then a single try.

run

Usage:

```
run <node> <slot>
run all <slot>
```

The `run` command executes the program in slot `<slot>` of the RCX at node `<node>` or of all connected RCX bricks, if the argument `all` is used.

stop

Usage:

```
stop <node>
stop all
```

The `stop` commands stops the execution of the program currently running on the RCX at node `<node>` or on all connected RCX bricks, if the argument `all` is used.

off

Usage:

```
off <node>
off all
```

The `off` command turns off the RCX brick at node `<node>` or all RCX bricks, if the argument `all` is used. There is no reply to this command.

signal

Usage:

```
signal <node> <sig>
signal all <sig>
```

The `signal` command sends the signal `<sig>` to the RCX brick at node `<node>` or to all connected RCX bricks, if the argument `all` is used. There is no reply to this command.

broadcast

Usage:

```
broadcast
broadcast <"on"|"off">
```

The `broadcast` commands turns the automatic signal broadcasting mechanism on or off, when used with a parameter, or displays the broadcasting settings. When automatic signal broadcasting is on, the console application will automatically broadcast all incoming signals to all connected RCX bricks other then the signal's sender.

III. RCX Information Retrieval

battery

Usage:

```
battery <node>
battery all
```

The `battery` command displays the voltage of the batteries of the RCX brick at node `<node>` or of all connected RCX bricks, if the argument `all` is used.

version

Usage:

```
version <node>
version all
```

The version command displays the version of the software of the RCX brick at node `<node>` or of all connected RCX bricks, if the argument `all` is used.

datalog

Usage:

```
datalog <node>
datalog all
```

The datalog command displays the datalog of the RCX brick at node `<node>` or of all connected RCX bricks, if the argument `all` is used. For further information on the datalog mechanism refer to the RCX documentation or to the NQC documentation.

get

Usage:

```
get <node> <res> <num>
get all <res> <num>
```

The `get` command displays the value of the instance `<num>` of resource `<res>` of the RCX brick at node `<node>` or of all connected RCX bricks, if the argument `all` is used. The valid resource specifiers and accepted arguments are:

Resource Number	Resource Description	Valid arguments (<num>)
0	variables	0-31
1	timers	0-3
3	motor state	0-2
8	currently selected program slot	0
9	formatted sensor value	0-2
10	sensor type	0-2
11	sensor mode	0-2
12	sensor raw value	0-2
13	boolean sensor value	0-2
14	minutes since start of RCX	0
15	last received message	0

IV. RCX Direct Control

beep

Usage:

```
beep <node> <tune>
beep all <tune>
```

The `beep` command plays the tune `<tune>` at the RCX brick at node `<node>` or at all connected RCX bricks, if the argument `all` is used. The valid tunes are:

Tune Number	Tune Description
0	Blip
1	Beep Beep
2	Downward Tones
3	Upward Tones
4	Low Buzz
5	Fast Upward Tones

sensor

Usage:

```
sensor <node> <num> <arg>
sensor all <num> <arg>
```

The `sensor` command changes the settings of the sensor number `<num>` of the RCX brick at node `<node>` or of all connected RCX bricks, if the argument `all` is used. The argument `<arg>` specifies the new type and mode of sensor. Setting the sensor type will also change its mode. The valid Sensor Arguments are:

Argument	Description
clear	set current value of to 0
Sensor Type Specifying Arguments	
unknown	no type set, raw mode
touch	touch sensor, bool mode
temperature	temperature sensor, celsius mode
light	light sensor, percent mode
rotation	rotation sensor, angle mode
Sensor Mode Specifying Arguments	
raw	value in 0..1023
bool	either 0 or 1
edge	sum of boolean transitions
puls	sum of boolean transitions divided by two
percent	raw value scaled to 0..100
celsius	1/10ths of a degree, -19.8..69.5
fahrenheit	1/10ths of a degree, -3.6..157.1
angle	1/16ths of a rotation

Example: `sensor 4 1 light raw clear` will set the type of sensor 1 of the RCX brick at node 4 to type 'light', mode 'raw' and set its current value to 0.

motor

Usage:

```
motor <node> <motor> <arg>
motor all <motor> <arg>
```

The `motor` command changes the power and direction of the motor outputs `<motor>` of the RCX brick at node `<node>` or at all connected RCX bricks, if the argument `all` is used. The argument `<motor>` is a combination of some or all of the characters ‘a’, ’b’ and ‘c’ in any order. The valid arguments `<arg>` are:

Argument	Description
on	turn motors on
off	turn motors off
float	allow motors to spin freely
fwd	set direction to forward
rev	set direction to reverse
flip	flip direction
1 - 7	set motor power to specified value

Example: `motor 4 ac float 4 flip on` will set motors A and C of RCX 4 to float mode, then set their power to 4, then flip their direction and finally turn them on.

V. Module Handling

list

Usage:

```
list
```

The `list` command lists all connected nodes with their addresses and all connected modules with their associated nodes to the console.

allow

Usage:

```
allow <node> <mod>
allow all <mod>
```

The `allow` command will, if not already existing, establish a connection to the file or named pipe `<mod>` and allow commands read from that connection to affect the RCX brick at node `<node>` or all RCX bricks if the argument `all` is used.

route

Usage:

```
route <node> <mod>
route all <mod>
```

The `route` command will, if not already existing, establish a connection to the file or named pipe `<mod>` and forward any information regarding the RCX brick at node `<node>` to the connection. If the argument `all` is used, then all information regarding any of the connected RCX bricks will be forwarded to the connection. For each newly routed node the connected module will receive a NODE message like the one the console displays when a new node has been found.

close

Usage:

```
close <node>
close <module>
close all
```

The `close` command closes the connection to a node `<node>` or to a module `<module>` or to all connected nodes and modules, if the argument `all` is used. Closing nodes does not change the permissions of modules associated with the node.