Providing an 'Interaction Interface' for networked robots using Squeak

Svein-Magnus Sørensen

June 21, 2004

Queensland University of Technology, Australia

Course: ITB844 Major Project
Completed: Semester 1, 2004

Supervisor: Ass. Prof. Joaquin Sitte
Faculty: Information Technology
Section: Smart Devices Laboratory

Avowal

I hereby declare that all work submitted as part of and along with this report is my own, except as acknowledged, and that it has not been previously submitted in part or full as a thesis or report to any other institution or company.

21. June, 2004.
Brisbane, Australia.

Svein-Magnus Sørensen
QUT Student # 04607058

Abstract

As the development of intelligent robotics move ahead, the development of Smart Appliances becomes increasingly more complex and expensive. Also the time-constraints involved in reading sensors are shorter, so the theory of the Agora Architecture was developed to allow pre-processing of input at the sensors and appliances themselves, instead of at a central server, and with transparent message passing. To promote re-use of the individual pieces a common interface for processing and interaction between the appliances had to be implemented. The Agora software implementation in Squeak was the first step towards a completely platform independent base on which smart appliances can be built.

Building upon the communications features of Agora, this project have started the development of the necessary Squeak classes required for a interface to transparently send and receive control-commands and responses to and from networked robots in a consistent way.

The project has also used the created interface to make an implementation for robots supporting the SerCom-protocol, focusing on K-Teams Koala-robot. As part of the project the provided Agora-implementation was modified to support the most recent version of Squeak. These modifications have been partly unsuccessful due to conflicts with Squeak-primitives, which can be fixed by a larger rewrite of Agora to use the newer Socket-implementation. Unfortunately the work involved in implementing this solution lies outside the scope of this project, but once it is completed the robot interface created will be fully operational.

Table of Contents

Page #

1 Introduction

1.1 Context - The Agora Architecture

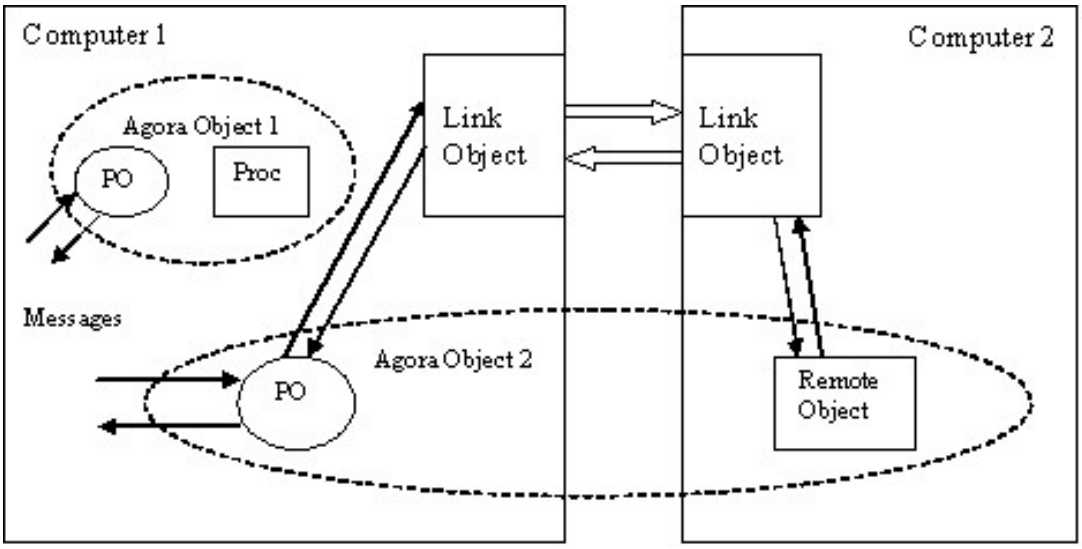
The Agora Architecture is an old idea to enable smart appliances to fulfil large and complex tasks on their own through cooperation with each other. The appliances are generally thought of as a range of embedded devices, each with a set of sensors and actuators. The special part about the Agora system is that each of the sensors will do its own pre-processing of the data it creates instead of just sending a raw feed to the main processor. This enables faster responses and less network traffic, while giving the option to code the controllers in a higher level language than is possible for normal hardware drivers. [1] [3]

Dr. Joaquin Sitte is one of the persons behind the development of the Agora system, his theories and ideas in part inspired by the Transputer technology created by Inmos Corporation. Transputers are small microcomputers with its own memory and set of point-to-point links to communicate with other Transputers, creating an easily modifiable distributed computing network. Despite the idea being ingenious the Transputers failed commercially and production was halted, but as the idea is even more relevant now than before, Dr. Sitte has been working to develop new and improved hardware and software to realize a working Agora system without the Transputers. The hardware is to be based on some of the theories behind the Transputers, but will be greatly improved and better suited for Agora than the original system ever was. In addition other advances in processing and networking that has occurred since the end of production can benefit the project greatly. [1] [3]

Despite having been conceived in 1994, the Agora Project is currently only in its beginning stages of development. Due to the hardware design and production phase taking longer than expected, the software development has been started on temporary devices awaiting its completion. An earlier project chose Squeak to be the development environment, partly due to its virtual machine architecture that makes any code created fully portable to any system capable of running the Squeak VM. This means almost all hardware for which there is a C-compiler available, including the new hardware under development and the BlueMod embedded processors on which much of the development currently happens. [2] [3]

One of these earlier projects is a working implementation of Agora in Squeak by Johannes Jansson. It has not implemented the complete system, but it has laid down the base functionality by providing transparent remote object handling over TCP-links. [2]

It is based upon this functionality that the work on this project begins.



The workings of the Agora System. Created by Johannes Jansson.

1.2 Project Description and Objectives

Building upon the communications features implemented in Agora by Johannes Jansson I were to create the base of a generic system for communicating controls and sensor-readings, and in other ways interact with, robots of various kinds.

This Interaction Interface was to be written in the Smalltalk language using the Squeak implementation of the Smalltalk environment. Other than this I was free to make any design decisions necessary to complete the interface. As a demonstration of its capabilities the project also required me to implement a version of the interface for use with the Koala-robot produced by K-Team Corporation that the Smart Devices Laboratory provided.

A generic robot control protocol with an implementation for controlling the Koala robot have already been written in Java by Erik Berglund at the Smart Devices Laboratory (SDL), and I were encouraged to use it as a theoretical base for the relevant parts of my interface.

As opposed to the Java implementation which sends commands to the robot and awaits it replies in the normal fashion, my implementation should make use of the remote object features of Agora to transparently execute calls to local proxy-objects on the actual objects existing on the robot itself.

Besides coding the interface and its implementation in Squeak, the project also required me to install Squeak on the Linux system running on a PC/104 embedded computer embedded on the Koala, and have Squeak communicate with the Koala-core over a serial link to access its motors and sensors.

In short, the following bulletpoints sum up my primary objectives:

- Create a generic robot interaction interface in Squeak that enables the passing and execution of messages to remote objects existing at a robot.
- Use the Agora-architecture implementation as a basis for the message passing and execution to remote objects over a TCP/IP network link.
- Install and run the Squeak VM on the Koala PC/104 system.
- Using the generic interface, implement and test a specific controller for use with the K-Team ’Koala’-robot owned by the Smart Devices Laboratory.

2 Environment

2.1 Smalltalk and Squeak

Smalltalk is a completely object oriented programming language and development environment that was thought up by Alan Kay in the 1960s. Smalltalk is the first pure object oriented environment created, meaning that everything is considered an object. The paradigm is based around passing messages with arguments to these objects, and having the object execute some code relevant to the message. Each of the messages understood by an object therefore has its own block of code that is executed when the message is received. [4]

It was originally invented as a language that would enable kids to learn programming in an easy and natural way, and still fulfils this role in a number of schools around the world. Due to this objective the syntax of Smalltalk is very easy to learn and grammatically similar to normal written English. For instance if one has some objects, 'Peter' 'theBall' and 'Lucy' and wish to pass a reference of 'theBall' between 'Peter' and 'Lucy', a natural way to code this would be: Peter pass: theBall to: Lucy.

If the operation returns a result this can easily be stored in any variable-object by the use of a assignment operator, either the colon-equals := as in Pascal or a underscore _ rendered as a arrow when used in a Smalltalk environment.

Due to its simple code and pure object orientated paradigm, coding in Smalltalk can usually get three times as much work done as the same number of lines in C or Java. [4] [6]

More information about Smalltalk itself can be found at [9].

Smalltalk has gone through a series of revisions and updates since its conception, and the currently most popular version is called Smalltalk-80 from the year it was finalized. There is a range of implementations of Smalltalk-80, most of them commercial versions, but there is one that stands out from the crowd by being free to download and use, namely Squeak.

Squeak is a complete development system that even includes applications for Web-browsing and chatting. All of the code the system is built from is included in the system through a construct called the Browser. From it you can view and change any of the code in the system and accept your changes for automatic recompilation and inclusion in the system on the fly with everything running! Of course this requires some self-constraint by a user as one could easily change vital code and make the system crash completely, but the benefits clearly outweigh the risk. The compile-protocol in Squeak is extremely-late-bound, so any code you develop in Squeak can be executed and inspected interactively as you go, without needing to go through a tiresome code → compile → test → debug cycle. If an error occurs you are presented with a debugger where you can go back a step, change the code an resume execution directly. Some people even write whole applications in the debugger just filling in code as required while they go along! [5]

As mentioned Squeak is free to download and use, as it comes with a very liberal licence. In fact, the only real demand placed on you by the licence is that anything you create in Squeak must be made available to the Squeak community in case someone else wants to improve or use what you have made. In return, you get the support of the entire community through its forums, swikis and mailing lists if there is anything you need help with. [8]

Squeak consists of two separate parts which is required for execution. Part one is the Virtual Machine that runs the Squeak code. The VM is written in Squeak itself, but through the use of a C-translator it can be ported to and compiled for almost any hardware for which a C-compiler is available. The second part is a Squeak image containing the system itself and all its applications. The image contains only Squeak-code and can be run bit-identical on any

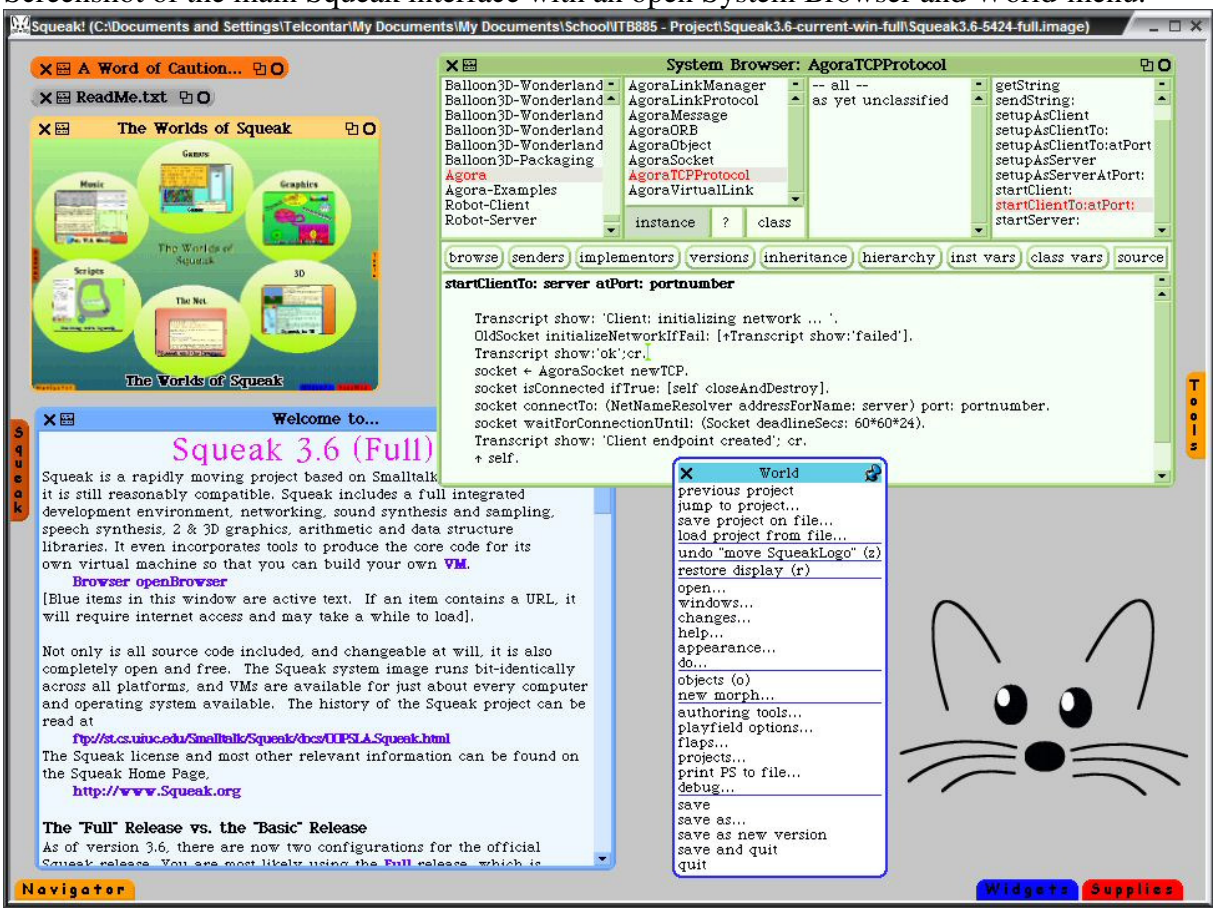
architecture for which a Squeak-VM has been ported. This means that anything you create in Squeak can be used globally without any changes required. In addition to this, running Squeak code is really fast with execution speeds comparable to programs written in C.

Many of the places where Squeak could potentially be used often have special hardware that needs separate drivers to work. Through the use of primitives written in C you even have the possibility to access these directly from the Squeak interface, so if the correct primitives are available you will never have to work with any other code than Smalltalk.

New applications and functionality does not exist as separate executable files, but are distributed as a source-code file that can be Filed-In to any Image and is automatically compiled there. You also have the option to File-Out any packages if you have created something and would like to share it with others without sending them your entire image. [5]

For some uses, like Agora or in mobile phones, the devices that will do the execution does not have large amounts of memory or processing power, so this puts tight constraints on anything that needs to run on the device itself. There are many projects attempting to shrink the Squeak image down as much as possible for uses like this, and despite the full Squeak image being in excess of 10 megabytes it is relatively easy to shrink it down to just a few megabytes by removing applications and classes that are not needed. The current size for those working with shrinking the image is currently just a few hundred KB. [2]

Screenshot of the main Squeak interface with an open System Browser and World-menu.



2.2 K-Team's Koala Robot

From the website of K-Team Corporation [10] about the Koala robot [11]: *

> Koala is a mid-size robot designed for real-world applications. Bigger than Khepera, more powerful, and capable of carrying larger accessories, Koala has the functionality necessary for use in practical applications (like sophisticated battery management), rides on 6 wheels for indoor all-terrain operation, and sports stylish bodywork for attractive demonstrations.

In addition to this, the robot retains the structure and is compatible with the smaller Khepera robot permitting programs and experiments to be migrated between the two. It also features 16 infra-red and ambient light sensors arranged around the robot in all directions, especially focused on the front.

The robot is constructed in a modular fashion, so a wide range of other sensors and extra utilities can be mounted to extend its functionality and range. Amongst these are ultrasonic sensors and LED range-finders. The robot-core supports serial-communication over a wired or wireless RS232 link following aschii sequences of the SerCom-protocol [12].

The version of the Koala robot that I used for testing is owned by the Smart Devices Laboratory, and has been extended with the following add-on modules:

- A optical pan-tilt turret-camera controlled from the Robot-base
- A PC/104 computer extension running Red Hat Linux. Se section 2.3 for more info.
- A wireless network link connected to the onboard Ethernet-port.

2.3 The PC/104 System

PC/104 is a low power computer system with a compact form factor following the IBM-PC specification. It was designed for use in embedded systems where the original PC-formfactor was unsuitable and is now approved as an IEEE standard. [14]

Since it is designed for embedded systems the formfactor is very compact with a self-stackable bus eliminating the need for backplanes and card cages. The connectors are rugged and powerful to handle potentially rough enviroments and a relaxed bus drives ensures low power requirements for extended battery-operation.

As the need for higher speeds have increased, a updated version of the standard called PC/104+ incorporates the newer PCI-bus into the old PC/104 form factor with full backwards compatibility. [15]

These are the specifications for the PC/104 extension featured on SDL's Koala-robot [15]:

- Pentium MMX 266mhz CPU and 64mb RAM
- Ultra Slim 20gb Hard-drive
- Onboard video, Ethernet, floppy, and input-controllers.

It is connected to the Koala-core through the RS232 serial-interface, and has a wireless-networking unit connected to the Ethernet-port. [13]

---

* Images of the Koala robot are property of the K-Team Corporation, Switzerland. K-Team are reponsible for design, manufacturing and distribution of the Koala and Khepera robots. The images are used in accordance with the terms stated on http://www.k-team.com/gallery/index.html - Accessed on June 18, 2004.

3 Development

3.1 The Squeak Environment

3.1.1 Installing Squeak

The first thing I needed to do when I started working on the project was getting Squeak itself up and running as it was the integral part I needed for everything. Through the Squeak website [8] it was easy to get binaries and images for both Windows and Linux.

The Windows binaries simply required unzipping and executing 'squeak.exe' to get up and running, but for the Linux on the PC/104 everything was not as simple. As Red-Hat Linux was installed on the computer I decided to try the easy way first by using the RPM package of the Squeak-VM. It failed at first, but after downloading the sources and image packages to complete the dependencies everything installed without any further complaints.

As there was no X-server installed on the PC/104 computer I decided to simply run Squeak in headless-mode without the graphical interface and it appeard to work nicely.
I could have gone to the trouble of installing a X-server, but as the Koala is a robot and not usually connected to a monitor, and were to be controlled remotely most of the time I decided against it.

3.1.2 Using Squeak

Learning to use Squeak was very simple. Guzdials book [5] were a great help in understanding the basics but for regular usage there is nothing in it that one could not figure out through a little trial and error.

With experience from a range of C-paradigm programming languages, Squeak came across as very alien and diffcult to work with at first. There is no obvious main area for putting the code belonging to a class, and the naming scheme appears illogical, however all of this quickly became clear after studying and using the system a bit. After doing this project the advantages of Squeak have proven themselves to me, and given me a new perspective on programming. When encountering problems where I see that Squeak can be useful compared to other languages I will start to favour the use of Squeak, and encourage others to do the same.

3.2 The Interaction Interface

3.2.1 Interface Design

After reviewing how the Squeak projects included in the image tend to be organized I decided to separate my work into two packages and work on each of them seperatly. One package for all of the server side classes, and one for the client side. This enables you to file-in only the Robot-Server or only the Robot-Client package if you require a small image footprint.

Here is a breakdown of the classes belonging to the Robot-Server package:

AgoraController

The primary component of the server functionality. It controls the Agora networking functionality and transparent message-passing between objects.

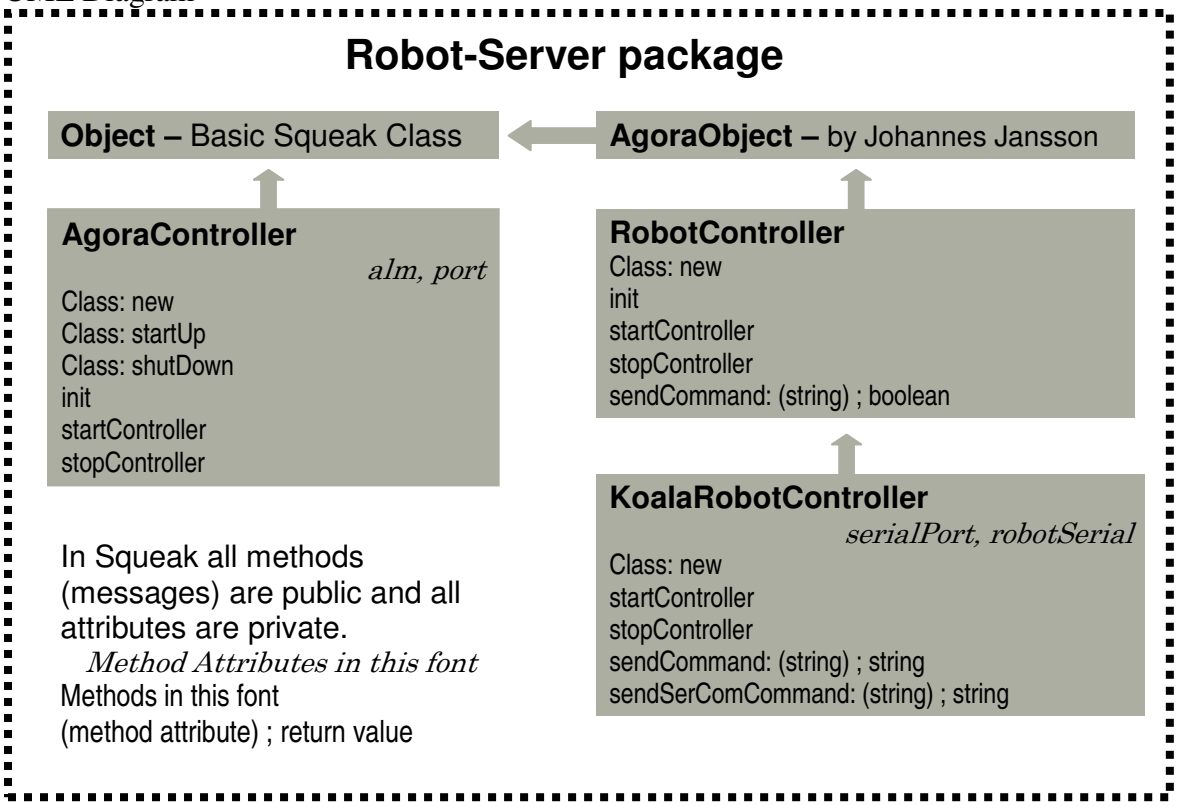
RobotController

An AgoraObject meant to be instantiated remotely by a connected client through the AgoraController. Besides for its setup-messages this superclass only understands the sendCommand message which is intended to pass a command on to the robot-core, and then return the result back to the message-sender.

KoalaRobotController

This is a specific implementation of the RobotController interface for use with the Koala robot. Its sendCommand message implements the sending of SerCom commands over the serial link between the PC/104 and the Koala-core. For simplicity it only relays the command on to the sendSerComCommand message which contains the functionality.

UML Diagram



And here is a breakdown of the classes belonging in the Robot-Client package:

RemoteClient

The primary component of the client functionality which performs the connection to a Agora-server on a user-specified robot. When connected it instantiates a RobotController object remotely on the server to handle the communication with the robot-core.
The RemoteClient provides interfaces for a series of generic commands for robot-control. It is a superclass and must be inherited by specific classes for each robot.
When deciding which generic commands to include for the system I relied heavily on the Robot controller in Java created by Erik Berglund.

SerComRemoteClient

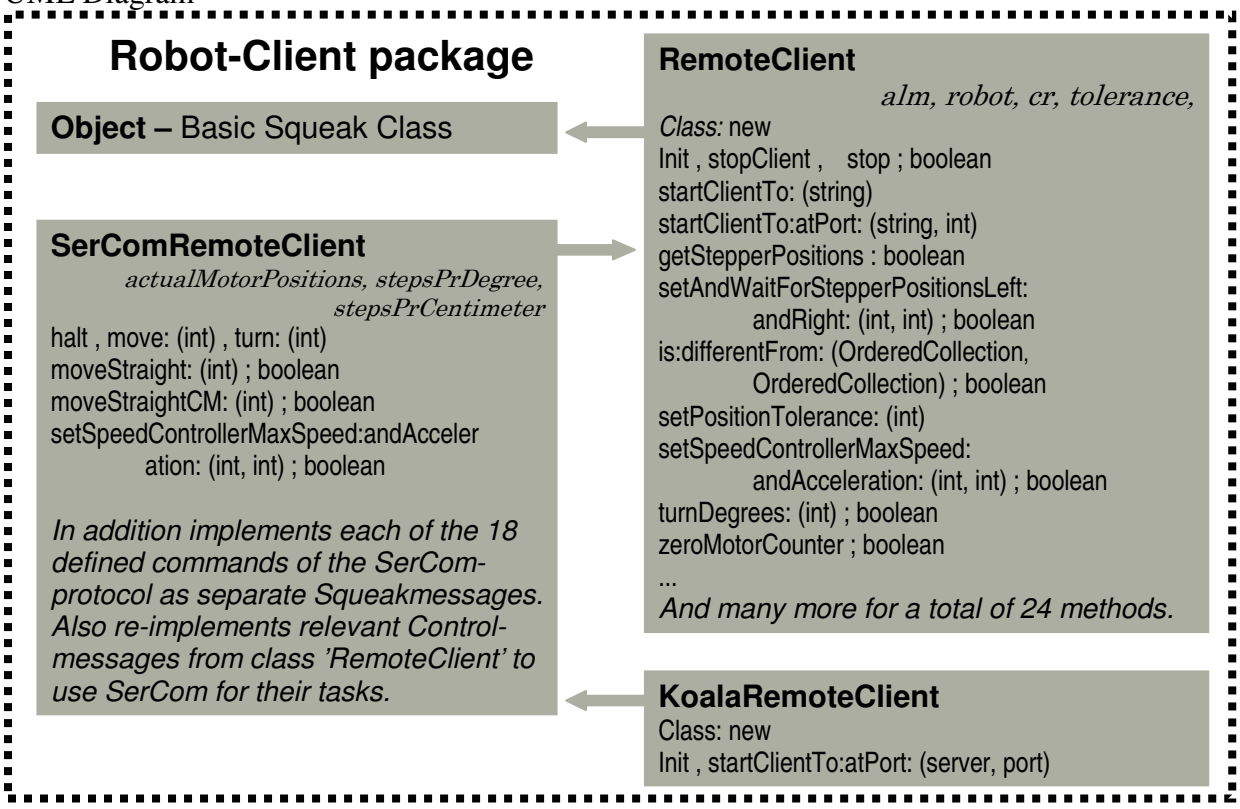
This is an implementation of the RemoteClient for robots understanding the SerCom-protocol. It extends the functionality of the RemoteClient and implements many of its default methods using SerCom specific control messages. It understands separate messages for each of the commands specified in the SerCom protocol and in addition implements some easy-to-use commands for manual robot control for testing and demonstration purposes.

KoalaRemoteClient

This is a simple subclass of the SerComRemoteClient that only specifies some initialization constants specific to the Koala robot. These constants must be set for each type of robot if the movement-controls are to function as expected.

..

UML Diagram



I have decided to locate the processing of sensor-input from the Koala in methods in the client side classes. This is not in line with the Agora theory of distributed processing at each seperate device, but the primary objective of my project was to create an Interaction Interface to communicate with robots through the remote object functionality of Agora, so to achieve this objective I decided this placement to be best suited for my project as it is similar to the solution used by the robot controller in Java that is currently in use.

Should it however in the future be required to have this functionality at the Robot-Server, the overhead of moving the relevant methods would be small. Internally they use a reference to the proxy-object when sending commands, and due to the transparent nature of Agora the methods need only be changed to use the real object on the Robot-Server instead and the transfer would be complete

.

3.2.2 Coding and Testing the Interface

After having decided on the main design-issues the amount on work required to code the system was not overwhelming. After the initial period of uncertainty about how to use the language Smalltalk turned out to be very efficient and easy to use. I progressed rapidly to a point where my interfaces and the Koala-implementation should be useable.

The interactive development cycle of Squeak allowed me to test small pieces of code at a time, ensuring that they worked before putting them into the relevant classes and connecting in the actual message sends. This makes me confident that the code I have written would work as expected during testing.

Initial testing quickly resulted in crashes, but through Squeaks built-in debugger the fault was located to be in the Agora-implementation I was making use of to handle network connections and transparent message passing. See section 3.3.1 for more info about this.

I was unsuccessful in resolving the problems with networking through Agora as the solutions available are unacceptable and/or appear to be too extensive and outside the scope and time-limits on my project. In place of a full test-cycle I have instead performed seperate testing of my methods through the Squeak debugger to ensure that they function as expected while awaiting a future update to the Agora-implementation. Once this update is in place and networking is working, the Interaction Interface should be fully operational.

While the sending of messages through Agora failed, communication between the Squeak-VM running at a client computer and the one running on the Koala robot was still indirectly verified through the crashing of both by failure of the socket-primitives when a connection attempt was made, however it is not useable for anything until a solution is ready.

3.3 Related Work

3.3.1 The Agora software-implementation

As a base to build my Interaction Interface I was provided with the Agora software-implementation created by Johannes Jansson. [2]

This implementation was only created as a small project to test the theories and does only include some very limited Agora functionality. By default Agora could only connect to another Squeak-VM running on localhost at port 54321. To be able to use the package for generic network connections I therefore had to write several extensions to allow for connections to custom-specified ports and servers.

Johannes Jansson created this software-implementation of Agora in Squeak version 3.5, implementing the remote object handling through the Squeak interface for TCP sockets. Unfortunately the implementation of the class Socket was completely rewritten during the change from Squeak version 3.5 to 3.6, and the old implementation renamed to OldSocket.

I only discovered this fact after initial testing, and had to perform a minor rewrite of several classes in the Agora package to use the backwards compatible OldSocket class. This solution is only temporary as the OldSocket class is deprecated and all code must be rewritten to use the new Socket class for the next version of Squeak when it might be removed.

After completing the rewrite of Agora I started a new series of testing where Squeak itself crashed instead of returning the debugger. I located the source of the crashes to be the Socket primitives handling the low-level connections with the network interface. Searches of the Squeak-dev e-mail archives [8] showed that the implementation of the OldSocket class have not been updated to handle its renaming. This results in new Socket objects being returned to messages from the Oldsocket class, which when passed unsupported parameters from the OldSocket class results in a crash.

There are three ways to resolve the situation with the TCP sockets, but the first two are unacceptable and the last is unfortunately outside the scope and time-limits of my project.

Solution 1 is to revert development on the Agora Project to Squeak 3.5. This is obviously a bad idea as the advantages of version 3.6 is highly sought after for qualities relevant to the Agora Project, as mentioned in [2] and [3].

Solution 2 is to perform a rewrite of the OldSocket class, resolving any dependencies where a new Socket is being returned and possibly recreating the primitives if they have been changed for the new Socket implementation. I strongly advise against using this solution as OldSocket is deprecated and might be removed completely in the next version of Squeak.

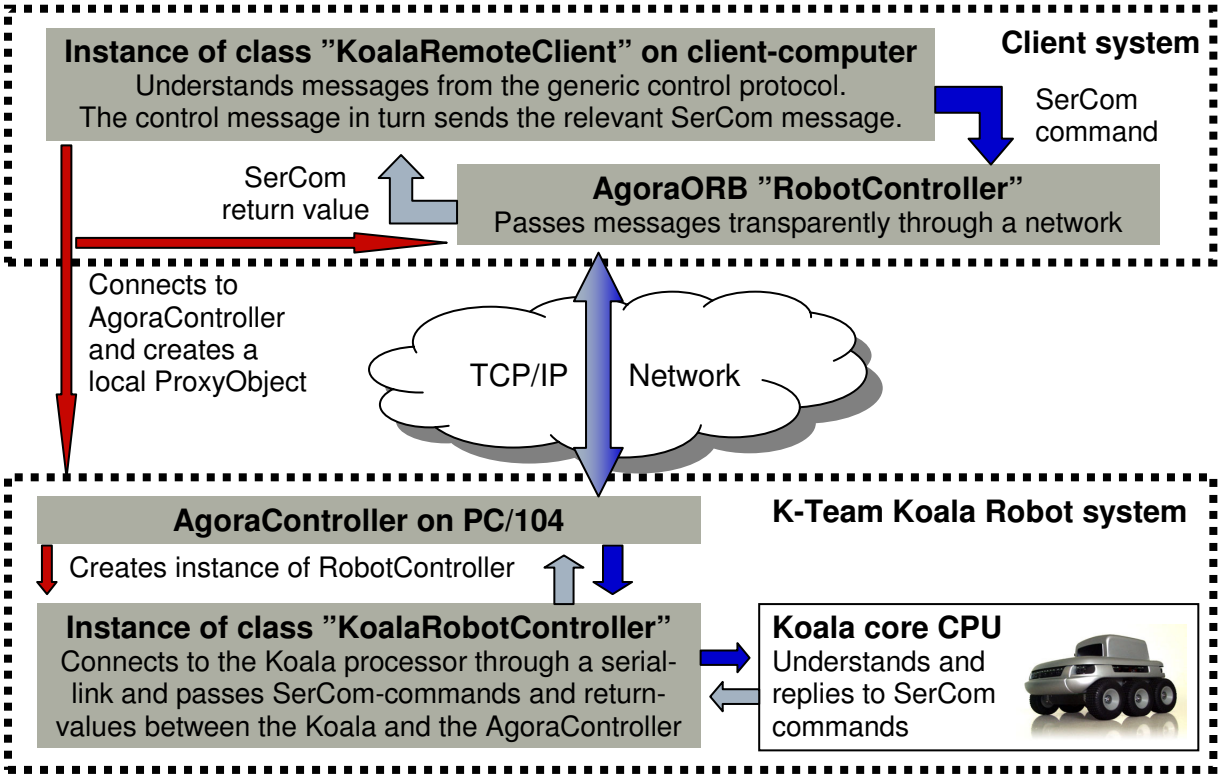
Solution 3 and the only viable long-term solution is to rewrite the Agora software-implementation to use the new Sockets. This involves a major rewrite of several of the Agora classes, and should also include a full analysis to ensure there are no other dependencies on deprecated features left in the Agora codebase.

4 Usage Documentation

4.1 How it all works together

1. A robot and a client-computer must both be running a Squeak-VM with the necessary Robot and Agora classes installed in the Squeak-Image.
2. An Agora-server must be started on the Squeak-VM on the robot to accept incoming connections from clients.
3. A client must instantiate and start a RemoteClient object on the local Squeak-VM.
4. The starting client tries to make an Agora-connection to the specified AgoraServer at the specified or default port (40000).
5. When a connection is established, the client tries to instantiate a RobotController on the remote VM through Agora, as well as creating a local ProxyObject.
6. Now the client can transparently send messages to, and receive return-values from, the ProxyObject like with any other local object, while the actual execution of the messages are taking place in the RobotController object on the Robot-Server.
   Any control-functions or sensor-values from the robot can now be accessed directly from the ProxyObject in a uniform manner, and the values returned to the client for processing.

Logical Flowchart



4.2 Using the Robot-classes

To use my robot interface you need start the Squeak virtual machines first.
On Windows this can be done by either double-clicking the 'squeak.exe' file, or dropping a Squeak-image onto the file itself to have it opened.
On Linux you need to run the Squeak binary with an image as an argument from the command line. To start the VM in headless mode, use the option "--headless".

As this is only a programming interface there is not yet any fancy graphical interfaces that you can use to control a robot. All messages must be sent by typing or pasting them into the Squeak workspace and selecting 'Do It' on each item as you wish them to be executed, or including them as parts of other methods.

When you have Squeak running on both computers you need to start the Agora-server on the robot. This can either be done manually if you have a graphical user interface, or be set up to run automatically on start-up for headless servers. Please refer to the class documentation in Squeak for information about starting the server automatically.
To start the Agora-server manually, copy the following code to its workspace and 'Do It'.

```
ac := AgoraController new.
ac startController.
```

Now a client should be able to connect to your server on the port specified in 'init'. The default port is 40000. The server can be shut down by sending the stopController message:

```
ac stopController.
```

Then you can start a RemoteClient on the remote object by doing the following code:

```
rc := KoalaRemoteClient new.
rc startClientTo: 'my.robots.name.or.address' atPort: 54321.
```

If you wish to use the default port (40000) the final argument "atPort: 54321" can be dropped. The RemoteClient will now try to establish a connection with the Robot server and create a KoalaRobotController object on the server, connected to a local proxy-object. When this is successful you can send any understood message directly to the 'rc' object, and it will be executed remotely on the robot and the answer returned through the regular Squeak interface. For example if you wish to tell the Koala to move 15 centimetres forward and turn 25 degrees to the left, simply do this code after the previous steps have been completed:

```
rc moveStraightCM: 15.
rc turnDegrees: -25.
```

For these two messages they a Boolean value is returned depending on the result. A more interesting command might be to read the robot speed at a given time and print it out.

```
| speed | "Sets a temporary variable called speed"
speed := rc readSpeed.
Transcript show: 'The robot is moving at ' + (speed/8) + 'mm/second'.
```

It is VERY important that you remember to stop the RemoteClient when you finish as the AgoraObjects will not be garbagecollected otherwise. This is a limitation from the underlying implementation of the Agora protocol. To stop the client simply do the following code:

```
rc stopClient.
```

For more information please refer to the Class comments inside Squeak.

5 Summary and Conclusions

The primary objective of this project has been completed by the creation of a Interaction Interface for networked robots in Squeak. The interface consists of the generic RobotServer and RobotClient interfaces which are both are extendable to fit many kinds of embedded devices and robots, or to be used further in the development of the Agora Project.

The Agora software-implementation has been used as a base for the transparent message passing between the RobotServer and RobotClient over a TCP/IP network. This however was partly unsuccessful due to major changes in the Squeak 3.6 Socket implementation. Several solutions to the problem has been suggested in section 3.3.1 but the implementation of these lies outside the scope and time-limits of this project.

However as all the pieces of the Interaction Interface has been tested during development through the Interactive debugging in Squeak, everything should be operational once the Socket-issues in the underlying Agora implementation has been resolved.

Squeak was successfully installed under Red Hat Linux on the Koala PC/104 extension. The graphical interface is not available as there is no X-server installed on the computer, but this is not a problem as the robot will be controlled remotely most of the time. Proper preparation of an image to autostart the required classes on a client computer before transferring the image to the Koala should ensure flawless operation in headless mode.

Based on the generic interfaces designed, I have created and tested a spesific implementation of the interfaces for a K-Team Koala-class robot, to be run on the PC/104 extension. Due to the problems described in sections 3.2.2 and 3.3.1, I was unable to perform actual passing of messages through Agora, so further testing on the robot could not be completed. The classes have instead been tested seperatly to the best of my ability and should operate successfully when the networking problems have been resolved.

As shown by the failures of the network Sockets in Squeak, the development model followed by the Squeak community results in that one can only guarantee that any given application will run as expected on the Squeak version that was used for development and testing of said application. This is a result of that the implementation of base classes can be changed completely between versions, without beeing required to provide a thoroughly tested backwards compatible alternative.

6 Future work

Concurrent operations
The Robot-Server package can be extended with many new classes to perform other operations concurrently with the control-suite implemented by me. Examples of such tasks can be time-critical operations like autonomous navigation and automatic collision detection and avoidance.

Robot-core access for concurrent tasks
To provide concurrent objects with direct access to the robot-core the control-commands implemented on the Client-side should be moved to the RobotController itself. This should prove to be a simple task as their internal implementation would simply use the real object on the robot instead of the proxy-object in the client and hence not require any changes.
This would be more in line with the Agora theory than handling signal processing at the client as I am currently doing, but as my project was

Manual GUI
As the interface I have created is only a programming interface it is difficult to use for direct manual control of a robot. If manual control of the robot is required a simple graphical interface for the RemoteClient should be created within Squeak to provide a human friendly way of sending its messages intuitively to a robot.

Higher level sensor output
The interface currently only provides basic returning of sensor-values as a OrderedCollection and raw-videofeeds. These outputs should be processed to be returned on a standard format according to the type of data, so that it can be more easily used by other devices or returned for human review in a manual GUI.

Acknowledgements

During my work with this project I have recieved aid and advice from many different peoples. Out of these I would especially like to thank...

Prof. Joaquin Sitte for his supervision and guidance, and for letting me do my bachelors project with him at the Smart Devices Laboratory.

Eirik Berglund for invaluable aid and good advice.

Roderick Taylor for helping out with resources and understanding Squeak.

Johannes Jansson for his work with the Agora implementation on which I based my work.

The Smart Devices Laboratory for providing me the resources I needed for my project.

References

Books and papers:

[1] Joaquin Sitte. *The Agora distributed processing architechure for robotics.* Report for "Proceedings of the 5th International Heinz-Nixdorf Symposium", *2001.*

[2] Johannes Jansson. *The Agora Software Implementation*. Masters Thesis, Smart Devices Laboratory, QUT Australia, 2003.

[3] Mathias Handorf. *Distributed Smalltalk with Bluetooth Wireless Links.* Bachelors Thesis, Smart Devices Laboratory, QUT Australia, 2003

[4] Alan Kay. *The early history of Smalltalk.* ACM SIGPLAN Notices, Volume 28 No. 3, 1993.

[5] Mark J. Guzdial. *Squeak: Object Oriented Design with Multimedia Applications*. Prentice Hall, 2001.

Websites, mailing-lists and online articles:

[6] Peter William Lount. A Brief Introduction to Smalltalk. [Accessed 8 June 2004] URL: http://www.smalltalk.org/#ABriefIntroduction

[7] Squeak-dev mailing list archive. [Accessed 10 June 2004] URL: http://lists.squeakfoundation.org/listinfo/squeak-dev

[8] Squeak Central. *URL:* http://www.squeak.org

[9] Smalltalk: The Pure Object Enviroment. *URL:* http://www.smalltalk.org

[10] K-Team Corporation. *URL:* http://www.k-team.com

[11] Koala Robot Information. [Accessed 15 June 2004] *URL:* http://www.k-team.com/robots/koala/

[12] SerCom protocol specification. [Accessed 15 June 2004] *URL:* http://www.k-team.com/software/sercom.html

[13] Koala PC/104 extension. [Accessed 15 June 2004] *URL:* http://www.k-team.com/robots/koala/pc104.html

[14] PC/104 Consortium. *URL:* http://www.pc104.org

[15] The PC/104 Technology. [Accessed 16 June 2004] *URL:* http://www.pc104.org/technology/

Appendix

Enclosed with this report is a CD-ROM containing all source-code changed and developed as part of this project, as well as a selection of other relevant files.

The contents of the CD-ROM can also be downloaded from this internet address:
http://www.menneske.org/projects/bachelorsproject/

Included on the CD-ROM is the following:

Folder:	**Contents:**
/	This report. Slides for the oral project presentation Slides to be assembled for the poster presentation.
/Articles/	A selection of articles relevant to the project.
/DevSqueak/	Squeak 3.6 for Win32 with Agora and Robot classes filed into Image.
/Pictures/	Pictures and graphics used in the report and on the slides.
/Source/	Filed out copies of the Agora, Robot-Server and Robot-Client package sources.
/Squeak3.6/	Win32 and Linux zip-archives of Squeak 3.6 binaries. Complete source code for Squeak 3.6. RedHat and Debian packages of the Squeak 3.6 Virtual Machine.