MOBILE CHEDAR MIDDLEWARE AND MOBILE P2P COMMUNICATION CENTER FOR NOKIA 770

by

André Mendes

Supervisor: **Research Student Mikko Vapa**

Special Assignment Report

20th of January 2006

Department of Mathematical Information Technology
University of Jyväskylä

Author: André Marques de Carvalho Mendes

Contact Information: Roninmäentie 1 G 23/a, 40500 Jyväskylä, Finland, amcmendes@gmail.com, +351919318927

Title: Mobile Chedar Middleware and Mobile P2P Communication Center for Nokia 770.

Work: Special Assignment Report

Number of Pages: 53

Study Program: Erasmus Exchange

Keywords: Peer-to-Peer, Middleware, Mobile Peer-to-Peer

ABSTRACT

This special assignment report presents the results of the Mobile Chedar Middleware and a Mobile Peer-to-Peer application for Nokia 770 mobile device development project. All the stages of the project were undertaken in University of Jyväskylä at Department of Mathematical Information Technology. The development of the Mobile Chedar Middleware and the Mobile P2P Communication Center aimed to determine the feasibility of Nokia 770 for communicating with Chedar nodes and mobile nodes in a P2P distributed environment. The experimental trials were done in a simulated environment for Nokia 770 Internet Tablet. A good performance was achieved in the simulator. The real Nokia 770 devices are not available yet, but because Nokia 770 is powered by Maemo with a Linux distribution, similar results are also expected from the real device.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

The author wishes to express sincere appreciation to research student Mikko Vapa for his assistance in the development of the project. In addition, special thanks to Niko Kotilainen whose familiarity with the area was helpful during the early programming phase of this undertaking. Thanks also to the staff of the Faculty of Information Technology for their support.

GLOSSARY

Bluetooth. A short-range radio technology that allows radio connections between devices within a 30-foot range of each other.

Breadth-First-Search. It is a tree search algorithm used for traversing or searching a tree, tree structure, or graph. Intuitively, you start at the root node and explore all the neighboring nodes. Then for each of those nodes, explore their unexplored neighbor nodes, and so on until the algorithm finds the goal.

Buffer. An amount of memory which temporally stores data to help compensate for differences in the transfer rate of data from one device to another.

Callback function. Function that is passed (by reference) to another functions. The other function calls the callback function under defined conditions (for instance, upon completion).

Chedar. Distributed computing and storage system that can be used as a platform for distributed Peer-to-Peer applications and Mobile Chedar as an extension to Chedar Network for mobile devices. [1]

CPU. Central Processing Unit – the module of the processor that controls and interprets the machine-language program and its execution.

GTK. Initially created for the graphics program GIMP, the GIMP Toolkit – abbreviated as GTK+ - is one of the most popular widget toolkits for the X Window System, intended for creating graphical user interfaces. GTK+ and Qt have supplanted Motif, previously the most widely-used X widget toolkit. [2]

Host. The name given to an individual computer running in the Internet.

Ip Address. Each machine connected to the Internet has an address known as an Internet Protocol address (IP address). The IP address takes the form of four numbers separated by dots, for instance: 123.45.67.890.

Maemo. It is a development platform for creating applications for the Nokia 770 Internet Tablet (http://www.nokia.com/770) and other maemo compliant handheld devices. [3]

Middleware. This term applies to a software layer that provides a programming abstraction as well as masks the heterogeneity of the underlying networks, hardware, operating systems and programming languages. [4]

Mobile Chedar. Mobile Peer-to-Peer middleware, which extends Chedar P2P Network for mobile devices. [1]

Multi-thread server. Contains a pool of threads which process requests. The server creates a new thread when a client makes a connection and destroys the thread when the client closes the connection.

Peer-to-Peer computer network. It is a network that relies on computing power at the edges (ends) of a communication rather than in the network itself. P2P networks are used for sharing content like audio, video or data in digital format and for sharing computation load between end machines.

PyGTK. It is a set of Python wrappers for the GTK GUI library. [2]

Python. It is an interpreted, interactive, object-oriented programming language. It is often compared to Tcl, Perl, Scheme, or Java. [5]

Selector. Performs an operation on a variety of data structures (classes) for example a list of sockets.

Server. A process that runs on a host that relays information to a client after processing the request of a client.

Socket. Provides an endpoint for a communication session. It is comprised of an IP address, a transport protocol and a port address.

Stream. A continuous flow of data, usually digitally encoded, designed to be processed sequentially.

TCP. Basic Unit of Transmission Control Protocol. It is the main transport protocol among Internet protocols, giving reliable and connection directional full duplex transmission stream. Data is delivered using the IP protocol.

Thread. Basic unit of program execution. It is a stream of computer instructions that is under control of a process.

Wireless LAN. Internet connection that can be accessed by radio waves. WLANs typically have a range of 50 – 100 m.

Chapter 1

INTRODUCTION

In Department of MIT there is a Peer-to-Peer research group (http://tisu.it.jyu.fi/cheesefactory/index.shtml) concentrating on distributed search of resources and their efficient use. This group has developed Chedar P2P System.

Chedar is a distributed computing and storage system that can be used as a platform for distributed peer-to-peer applications and Mobile Chedar is an extension to Chedar network for mobile devices. Chedar is Java-based peer-to-peer computing platform that can be used to build a network of workstations where each node is providing and consuming resources. Chedar has been designed as a general platform for locating resources meaning that resource types can vary and be customized for different user needs. Chedar is built upon TCP-sockets and XML technologies. [1]

The aim of this project was to develop a Mobile Chedar Middleware and a Mobile Peer-to-Peer application for the Nokia 770 mobile device. Python programming language was selected for the development. The Maemo Platform (http://maemo.org) [3] was used to create a simulated environment of Nokia 770 Internet Tablet to run the Python programs. The main purpose of this development is getting an extension of Chedar system to the Nokia 770 mobile device (http://www.nokia.com/770).

Mobile Chedar has already been implemented for Nokia 6600 using Bluetooth radio technology. [6] However Nokia 770 supports Wireless LAN and allows creating multimedia P2P applications (video & music) with large screen size.

This report is structured in these chapters. Chapter 2 contains the relevant issues about Python Programming Language. Chapter 3 contains an overview of the Mobile Chedar Middleware and the Mobile P2P Communication Center. Chapter 4 contains a description of the Chedar protocol and Group Chat protocol. Chapter 5 contains a specification of the UML Class Diagram, Classes and Methods. Chapter 6 contains the tests used to assure the software reliability. Chapter 7 contains some suggestions for future applications. Chapter 8 contains the amount of working hours. Chapter 9 contains conclusions about the developed project.

Chapter 2

PYTHON STUDY

Python Programming Language was selected for the project development. It was created in the early 1990s by Guido van Rossum at Stichting Mathematisch Centrum (CWI) in the Netherlands as a successor of the ABC programming language.

Python is a multi-paradigm language. This means that, rather than forcing programmers to adopt one particular style of coding, it permits several: object orientation, structured programming, functional programming, aspect-oriented programming, and more recently, design by contract, are all supported. Python is dynamically type-checked and uses garbage collection for memory management. An important feature of Python is dynamic name resolution, which binds method and variable names during program execution. The efficient high-level data structures and a simple but effective approach to object-oriented programming is one of the reasons that explains why Python has gained so wide acceptance. It is also important to mention that many Python interpreters have been ported (i.e. changed to make it work on) to many platforms. All Python programs can work on all these platforms without requiring changes.

However, Python is sometimes classified as a "scripting programming language". Python proponents prefer to call it a high level dynamic programming language, on the grounds that "scripting language" implies a language that is only used for simple shell scripts or that refers to a language like JavaScript: much simpler and, for most purposes, less capable than "real" programming languages such as

Python. For example Google uses Python for many tasks including the backends of web apps such as Gmail [7] and Google Maps [8] and for many of its search-engine internals. [2]

Another important goal of the language is ease of extensibility. New built-in modules are easily written in C or C++. Python can also be used as an extension language for existing modules and applications that need a programmable interface.

Python's support for object oriented programming paradigm is vast. It supports polymorphism, not only within a class hierarchy but also by duck typing. Any object can be used for any type, and it will work as long as it has the proper methods and attributes. And everything in Python is an object, including classes, functions, numbers and modules. Python also has support for metaclasses, an advanced tool for enhancing classes' functionality. Naturally, inheritance, including multiple inheritances, is supported. Python also has some features that make it possible to write large programs, even though it lacks most forms of compile-time checking: a program can be constructed out of a number of modules, each of which defines its own namespace, and modules can define classes which provide further encapsulation. Exception handling makes it possible to catch errors where required without cluttering all code with error checking.

Python also provides facilities for introspection, so that a debugger or profiler (or other development tools) for Python programs can be written in Python itself. There is also a generic way to convert an object into a stream bytes and back, which can be used to implement persistent objects as well as various distributed objects.

Chapter 3

OVERVIEW

Mobile Chedar Middleware

The first phase of the project was to develop a middleware for the Nokia 770 mobile device and test how the middleware works using a simple application running in a simulated mobile node. The developed middleware provides methods to send queries, receive queries, forward queries and process requests. All these functionalities were tested with an application performing video file requests.

Following picture presents communication among Chedar P2P Network and Nokia 770 mobile devices, as well the locations of Mobile Chedar middleware and application.

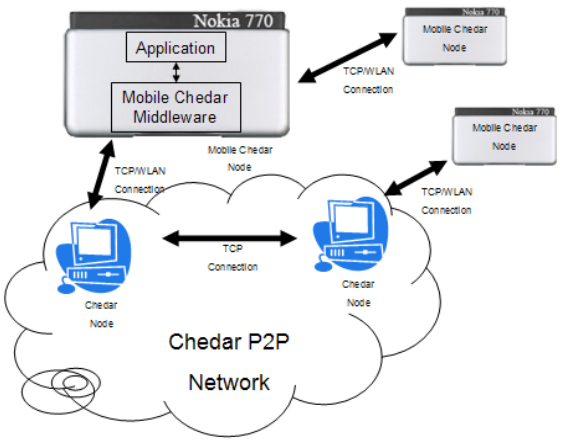

Figure 1 – Communication among Chedar P2P Network and Nokia 770 tablets.

Chapter 3 – Overview

Mobile P2P Communication Center

The next phase of the project was to develop a Mobile Peer-to-Peer application for group communication in real-time. This application can be used for example by students to subscribe a specific lecture. All the students joined in the same lecture stream can read and add information to it. Modifications to the stream will be visible to the lecture group. The developed application provides user functionalities to locate stream, join stream, publish stream, update stream and close stream.

Following pictures present communication between two users. The Special Assignment stream is created by the user 1. The user 2 joins to the new stream and starts to see in real-time what is being added. He is also adding by himself new information.

Figure 2 – User 1 created a "Special Assignment" stream and added text on it.

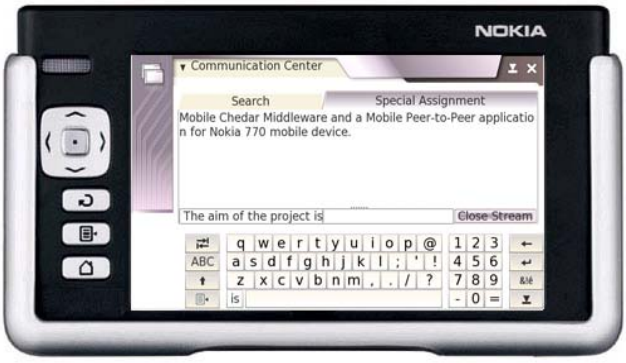

Figure 3 – User 2 joined to the "Special Assignment" stream and adds text on it.

Chapter 4

PROTOCOLS

Chedar Protocol

There are three types of messages used for topology management in Chedar. When one node wants to establish a new connection to some node it sends the standard message "CHEDARPORT<port of the node>". After this step both nodes are ready to communicate. This communication is based on two types of messages. Nodes can send messages with queries and messages with replies to the received queries. Following pictures present the three types of messages.

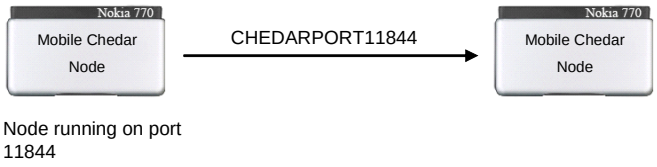

Figure 4 – Standard message to establish a connection.

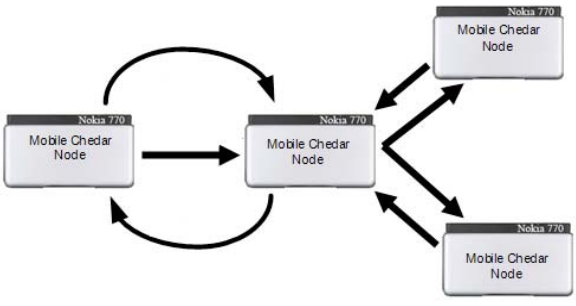

Figure 5 – Message exchanges.

Group Chat Protocol

There are two types of messages used in the Group Chat Protocol. When one node wants to join to a specific stream it sends a message with: "JOIN <name of the stream>". If one node wants to send an update it sends a message with: "UPDATE <text to be added>". Following pictures present the two types of messages.

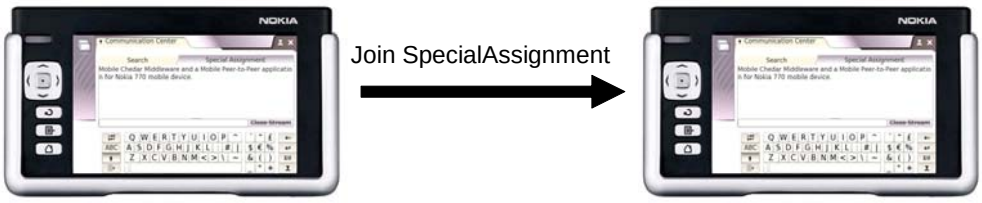

Figure 6 – Join message.

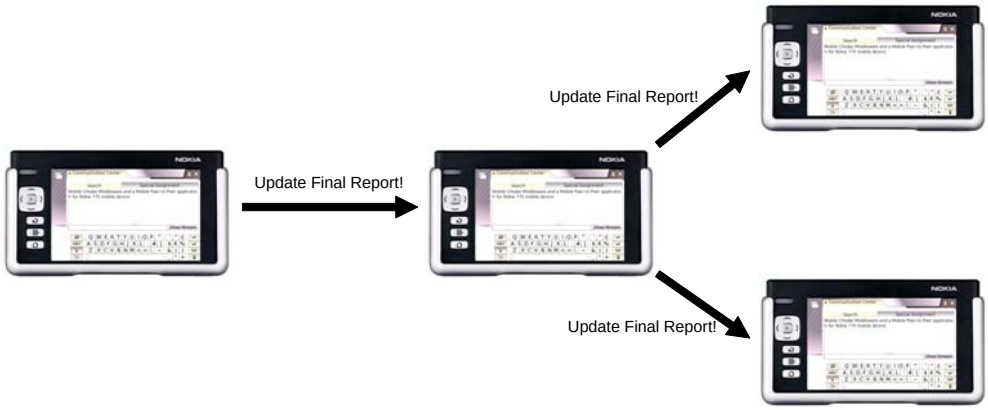

Figure 7 – Update message. The update will be forwarded to all the nodes that have subscribed to the same stream.

Chapter 5

CLASSES AND METHODS

UML Class Diagram

Figure 8 illustrates the UML class diagram of classes in MP2P Communication Center and Mobile Chedar.

Everything is an object in Python. However the UML class diagram only has the developed main classes. In this abstraction level the objects provided by Python were not shown in this diagram. In the next subchapters all the data structures will be shown and explained.

Mobile Chedar Middleware for Nokia 770

Mobile Chedar Middleware is a communication layer that allows applications to discover resources of Chedar P2P network on a mobile device. This section describes the classes and methods of Mobile Chedar for Nokia 770 implementation. Figure 9 illustrates the position of the Middleware among layers of software and hardware.

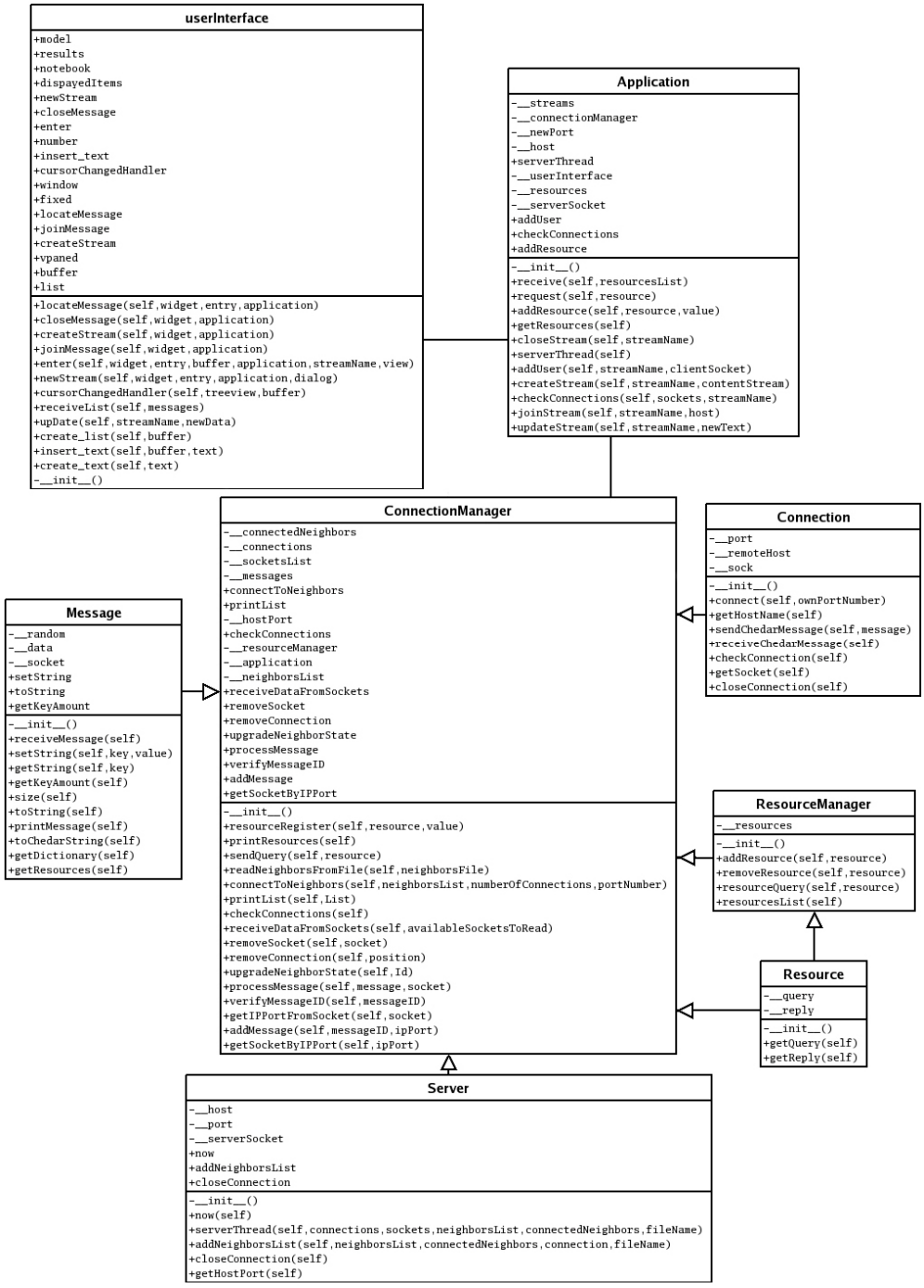

Figure 8 - UML class diagram of the developed software.

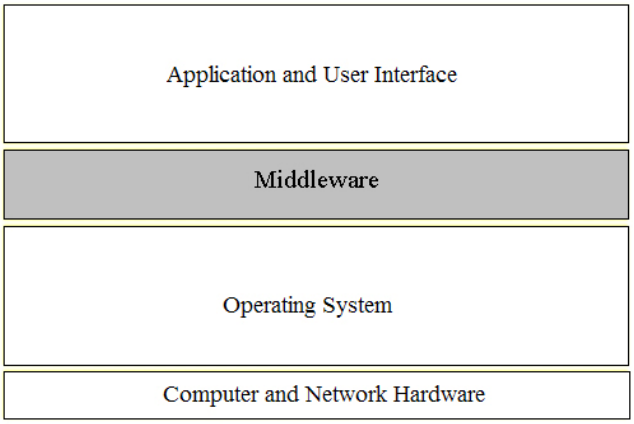

Figure 9 – Location of the Middleware in the structure of the software and the hardware.

Class Message

This class is used to save one or more queries. There are two types of queries: requests and replies. All the messages will be exchanged among nodes. Figure 10 illustrates the fields of the message.

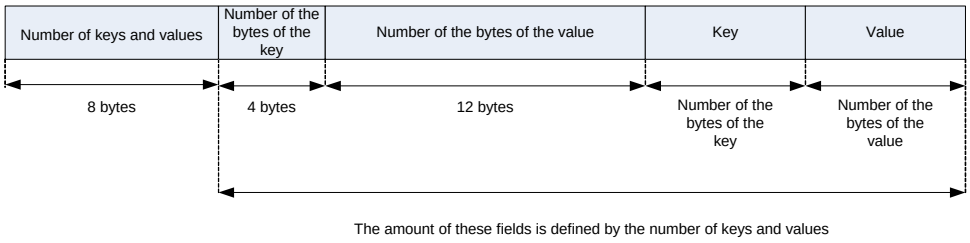

Figure 10 – Fields of the message.

The Figure 11 illustrates an example of a message with two fields.

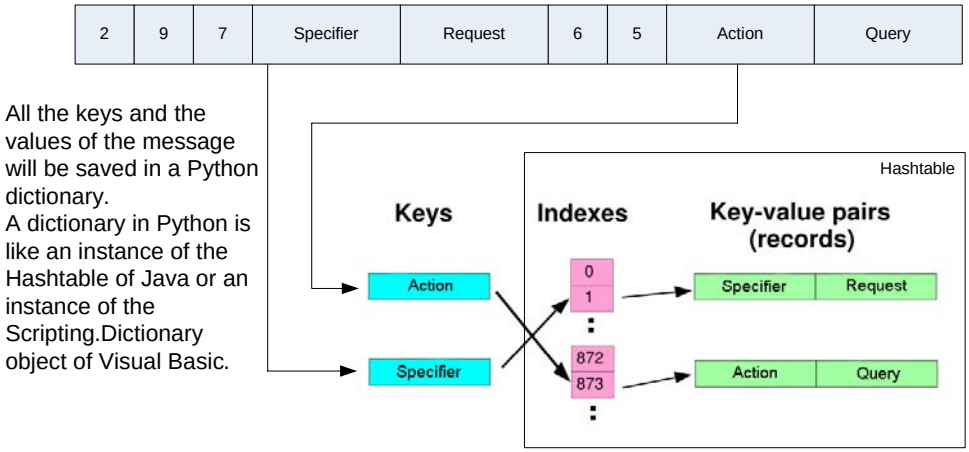

Figure 11 – Message example and hashtable data structure.

Next the methods of the Message class will be described.

`__init__(self, socket = None)`

This is the message constructor. If the socket is not specified an empty message with a random id will be created. Otherwise the received socket will be assigned to a private variable and it will be later read to receive the new message.

`receiveMessage(self)`

The contents of the message will be received through the specified socket. In the beginning the number of keys and values (8 bytes) will be read. After this operation the number of the bytes of the key (4 bytes) and the number of the bytes of the value (12 bytes) will be read. Finally the key and the value will be read and both will be saved in the message.

`setString(self, key, value)`
Add the `key` and the `value` to the message.

`getString(self, key)`
Return the string value associated to the specified `key`.

`getKeyAmount(self)`
Return the number of keys of the message.

`size(self)`
Return the number of characters of all values.

Class Connection

This class is used to create a TCP connection. Each node has different TCP connections for all of its neighbors. The Figure 12 illustrates the data structure of the connection.

Figure 12 – Data structure of the connection.

Next the methods of the Connection class will be described.

`__init__(self, Id = None, sock = None)`
This is the connection constructor. The `Id` has information about the host and the port (<host>:<port>). If the `Id` is specified, the host and the port will be assigned to private variables. These will be used to establish a TCP connection. Otherwise the socket will be specified through the variable `sock` and assigned to a private variable. Then the TCP connection will be created with this socket.

`connect(self, ownPortNumber)`
Connect to the host and the port given by the private variables. After this step a standard protocol message (CHEDARPORT<`ownPortNumber`>) will be sent through the socket. The node is always listening for new connections in the specified port number.

`getHostName(self)`
Return the Id of the connection. The Id is composed of port and host (<host:port>).

`sendChedarMessage(self, message)`
Send the specified `message` through the socket. The `message` is formatted according to the rules defined by the protocol.

`receiveChedarMessage(self)`
Receive the standard Chedar message through the socket.

`checkConnection(self)`
Receive 15 bytes of data and verify if they match with the expected bytes defined by the protocol. The connection will be consider valid if the expected bytes will be received otherwise it will be consider invalid and it will be dropped.

`getSocket(self)`
Return the socket of the connection.

`closeConnection(self)`
Close the socket of the connection.

Class Resource

This class is used to create a resource. One resource is defined by its name and its location. The Figure 13 illustrates the data structure of the resource.

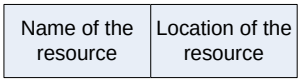

Figure 13 – Data structure of the resource.

Next the methods of the Resource class will be described.

`__init__(self, query, reply)`
This is the resource constructor. The specified `query` and the specified `reply` will be assigned to private variables.

`getQuery(self)`
Return the name of the resource.

`getReply(self)`
Return the location of the resource.

Class Resource Manager

This class is used to manage resources database. This searchable database allows resources to be added and removed.

The Figure 14 illustrates the data structure of the resource manager.

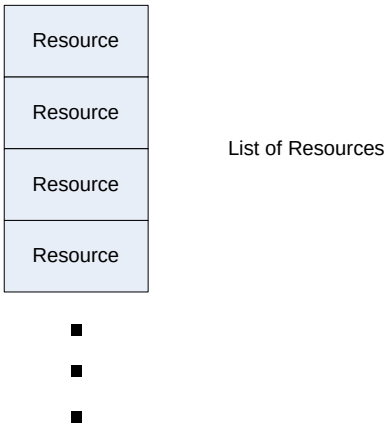

Figure 14 – Data structure of the resource manager.

Next the methods of the ResourceManager class will be described.

`__init__(self)`
This is the resource manager constructor. An empty list will be created to save the resources.

`addResource(self, resource)`
Add the specified `resource` to resources database.

`removeResource(self, resource)`
Remove `resource` from resources database.

`resourceQuery(self, resource)`
The search result matching the `resource` in resources database will be returned in a list. The criterion for matching the query is the specified `resource` name.

`resourcesList(self)`
Print available resources.

Chapter 5 – Classes and Methods

Class Server

This class is used to accept new TCP connections. It is a multi-thread server with one thread per new TCP connection. The Figure 15 illustrates the data structure of the server.

Figure 15 – Data structure of the server.

Next the methods of the Server class will be described.

`__init__(self, host = None, port = None)`
This is the server constructor. If the `host` and `port` are not specified the server will start running on the default host "localhost" and port 11844. Otherwise it will start to run in the given `host` and `port`.

`now(self)`
Return the current time.

`serverThread(self, connections, sockets, neighborsList, connectedNeighbors, fileName)`
Listen for new TCP connections and add them to `connections` database. The sockets of the connections will be added to `sockets` database.

`addNeighborsList(self, neighborsList, connectedNeighbors, connection, fileName)`
If the Id (<host:port>) is unknown it will be added to `neighborsList` database. Otherwise the position database of the neighbor will be returned. This position will be used to determine the new connected neighbor.

`closeConnection(self)`
Close the socket of the server.

`getHostPort(self)`
Return the Id (<host:port>) of the server socket.

Class ConnectionManager

This class is used to manage connections, to process requests, to receive and send queries. The established connections are always being verified by a thread. The available data to read from the connections will be immediately processed. The queries can be sent, received and forwarded. ConnectionManager will provide all the methods of the middleware. The Figures 16 and 17 illustrate the data structures of the connection manager.

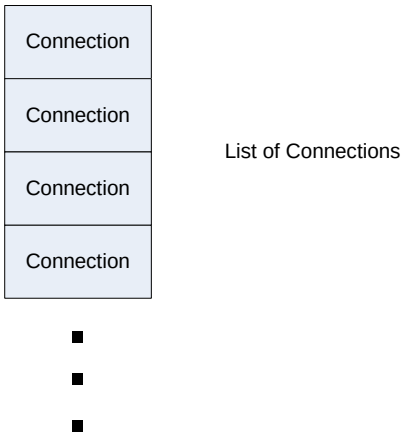

Figure 16 – List of Connections. Data structures of the connection manager.

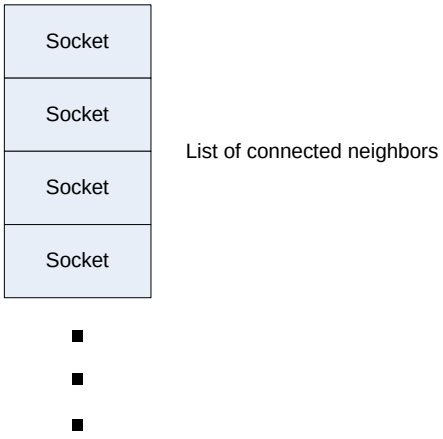

Figure 17 – List of connected neighbors. Data structures of the connection manager.

Next the methods of the ConnectionManager class will be described.

`__init__(self, neighborsFile, numberOfConnections, portNumber, application)`
This is the connection manager constructor. The connections database, the sockets database, the messages database, the connected neighbors database and a thread to check connections will be created.

`resourceRegister(self, resource, value)`
Add a `resource` to resources database.

`printResources(self)`
Print resources.

`sendQuery(self, resource)`
Send a query to connected neighbors to look for the specified `resource` name.

`readNeighborsFromFile(self, neighborsFile)`
Read the Ids from the `neighborsFile` to neighbors database.

`ConnectToNeighbors(self, neighborsList, numberOfConnections, portNumber)`
Establish a specified number of TCP connections with own neighbors.

`printList(self, List)`
Print the list.

`checkConnections(self)`
The selector is used to check if there are sockets with data to read. The available data to read will be received.

`receiveDataFromSockets(self, availableSocketsToRead)`
Receive data from sockets. The received messages will be processed. If some messages are not properly received the connection and the socket will be removed and a new connection will be established to keep the amount of connections.

`removeSocket(self, socket)`
Remove the `socket` and return the position of the removed socket.

`removeConnection(self, position)`
Remove the connection at the specified `position` and return its Id.

`upgradeNeighborState(self, Id)`
Delete the specified `Id` from connected neighbors database. Add the specified `Id` to neighbors database.

`processMessage(self, message, socket)`
Verify if the query is a request or a reply. Send a message reply if there is at least one available resource that matches with the request. The time to live (ttl) will be always decreased by one. If ttl is zero after being decreased, the message will be dropped. Otherwise the query will be forwarded to all neighbors if resource is found and the time to live (ttl) is still greater than zero.

`verifyMessageId(self, messageId, ipPort)`
Verify if the list of messages has the specified `messageId`. The position of the `messageId` will be returned if the `messageId` will be found. This method is used to drop repeated messages and prevent cycles among nodes.

`getIpPortFromSocket(self, socket)`
Return the host name of the specified `socket`.

`addMessage(self, messageId, ipPort)`
Add the pair (<`messageId`>, <`ipPort`>) to messages database. The messages database will keep the maximum amount of fifty pairs. The first elements will be replaced after that.

`getSocketByIpPort(self, ipPort)`
Return the socket with the specified `ipPort`.

Chapter 5 – Classes and Methods

MP2P Communication Center
Application for Nokia 770

The purpose of MP2P Communication Center application is to provide real-time chat communication among multiple users. This application can be used by the students to subscribe to a specific lecture course. All the students joined in the same lecture course can read and add information to it. Lecture modifications will be visible to the lecture group.

Figure 18 illustrates the position of the Application among layers of software and hardware.

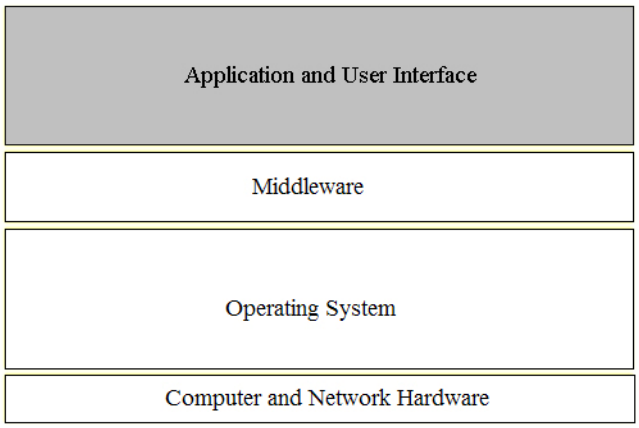

Figure 18 - Location of the Application in the structure of the software and the hardware.

Class Application

This class uses Mobile Chedar Middleware for Nokia 770 to provide the following functionalities to the users: locate stream, join stream, publish stream, update stream and close stream. The Figure 19 illustrates the data structures of the application.

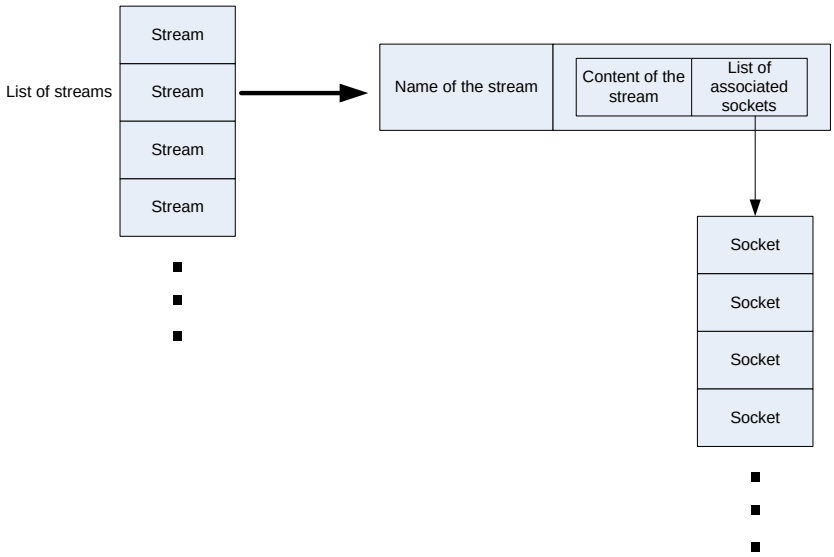

Figure 19 – Data structures of the application.

Next the methods of the Application class will be described.

`__init__(self, filename, interface)`
This is the application constructor. A list of streams will be created. Each element of this list is a pair. The first element of the pair is the name of the stream and the second element is a hierarchic pair. The first element of this hierarchic pair is the content of the stream and the second element is a list of associated sockets. The

Connection Manager will be instantiated and a multi-thread application server will be created to listen for new connections.

`receive(self, resourcesList)`
Receive the `resourcesList` provided by the middleware.

`request(self, resource)`
Call the available method of Middleware to request neighbors about the specified `resource`.

`addResource(self, resource, value)`
Call the available method of Middleware to add the specified `resource` with the specified `value`.

`getResources(self)`
Call the available method of Middleware to print resources database.

`closeStream(self, streamName)`
The sockets associated to the specified `streamName` will be closed and removed.

`serverThread(self)`
Listen for new TCP connections. Available data will be received through the new sockets. The new user will be added if received data matches with expected bytes defined by the protocol.

`addUser(self, streamName, clientSocket)`
The client socket will be added to streams database. A thread to check available data to read sockets of the specified stream will be created.

`createStream(self, streamName, contentStream)`
A new stream with the `streamName` will be created. The blank spaces will be replaced by underscores. The stream location will be created according to the

rules defined by the protocol. Finally call the available method of Middleware to add this new resource, and add the new stream to streams database.

`checkConnections(self, sockets, streamName)`
The selector is used to check if there are `sockets` with data to read. The available data to read will be received. An update will be done on streams database and sent to the user interface if the first seven bytes match with expected bytes defined by the protocol. Otherwise the socket will be removed from streams database.

`joinStream(self, streamName, host)`
The specified `host` will be parsed to get the ip and port. A new TCP connection to these ip and port will be established. A join message defined by the protocol will be sent through the new socket. This socket will be added to the streams database. The available method of Middleware will be called to add the resource, and the stream to streams database. A thread to check available data to read sockets of this new specified stream will be created.

`updateStream(self, streamName, newText)`
The new specified text will be added to the content of the specified stream and updated on streams database. An update message defined by the streaming protocol will be sent through the associated sockets.

MP2P Communication Center User Interface for Nokia 770

The user interface contains the elements of the computer screen that user interacts with. This includes visual appearance, icons, navigational elements and request for text.

Figure 20 illustrates the position of the User Interface among layers of software and hardware.

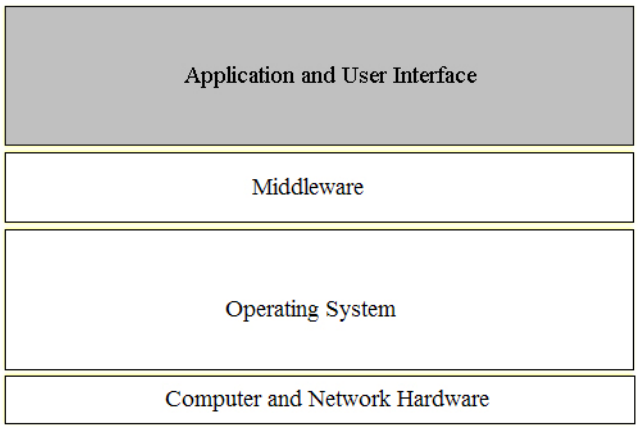

Figure 20 - Location of the User Interface in the structure of the software and the hardware.

Class User Interface

This class is used to create all windows, buttons and dialog boxes. The next figures show the application running on Nokia 770 and its representation with GTK objects. [9]

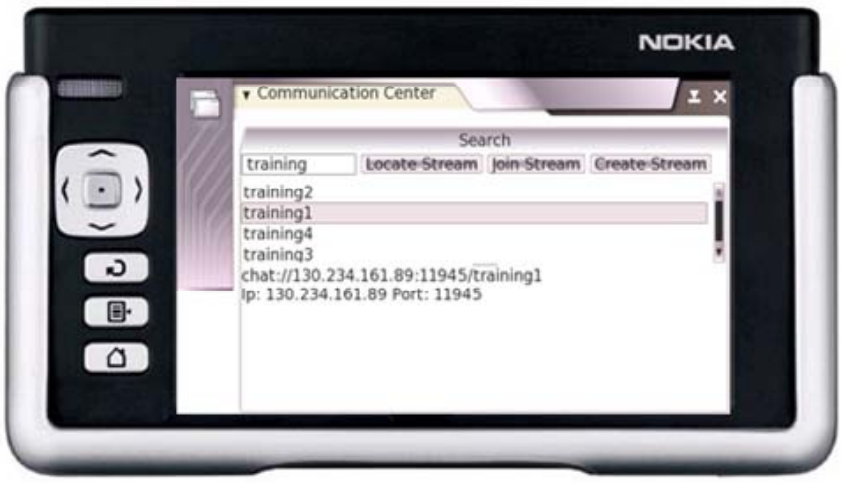

Figure 21 – Application running in Nokia 770.

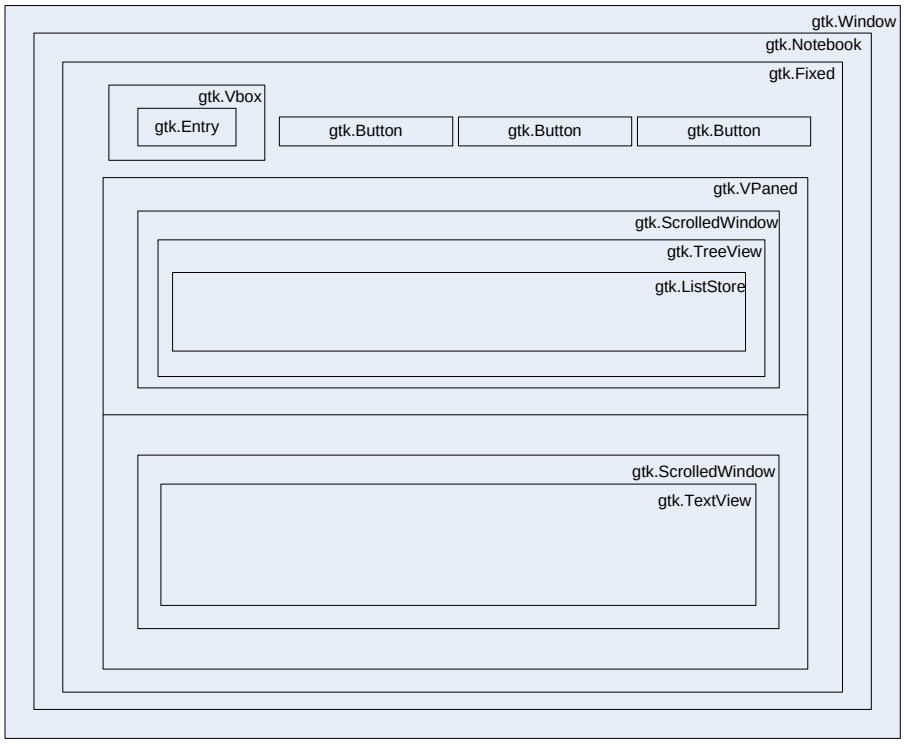

Figure 22 – Representation of the search view with GTK objects.

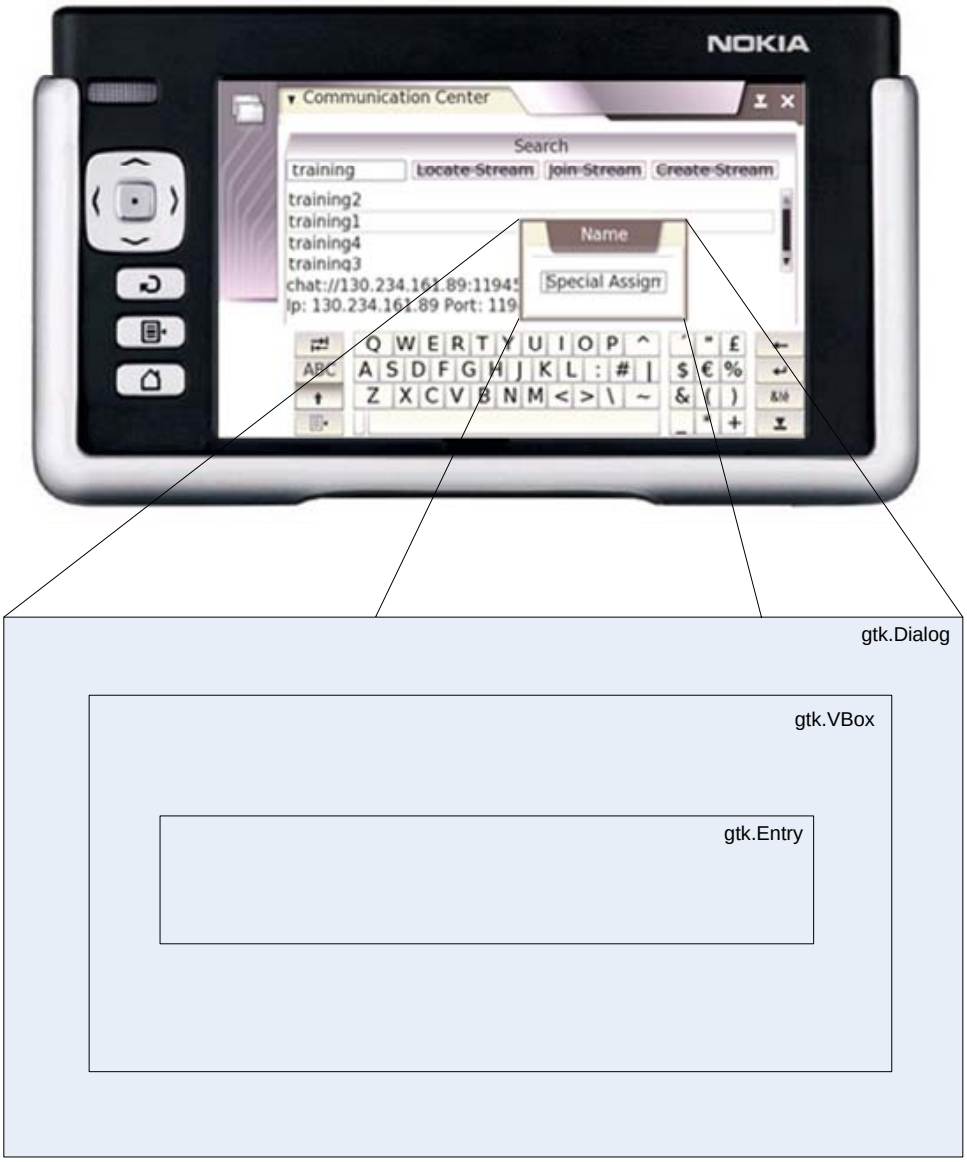

Figure 23 – Representation of the create stream dialog with GTK objects.

Figure 24 – Application running in Nokia 770.

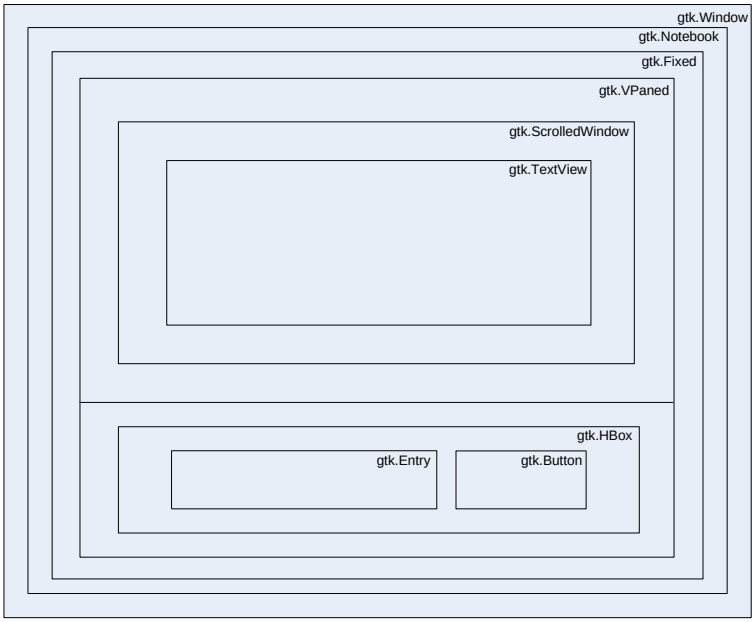

Figure 25 – Representation of the stream view with GTK objects.

Next the methods of the User Interface class will be described.

`__init__(self)`
This is the user interface constructor. This method is used to create a new window, a notebook, a fixed container, a vertical box, a single text line entry field, a text view, a scrolled window and the buttons. An empty list will be created to save the displayed items. Each element of this list is a triple. The first element of the triple is the name of the stream. The second element of the list is a buffer. This buffer contains the displayed text. The third element is a text view. This text view will display the text contained in the buffer.

Callback Functions

Callback functions are passed (by reference) to another function. The other functions call the callback functions under defined conditions (for instance navigation in the scroll window, a button click, etc.) The Callback functions are displayed below.

`locateMessage(self, widget, entry, application)`
Call the available method in the `application` to request the stream specified in the text line `entry` field. Create an empty list to save the request results.

`closeMessage(self, widget, application)`
Call the available method in the `application` to close the displayed stream. The displayed tab will be closed.

`createStream(self, widget, application)`
Create a new dialog box to write the name of the stream.

`joinMessage(self, widget, application)`
Call the available method in the `application` to join to a stream. Create a fixed container and append it to the notebook. Create a container with two panes arranged vertically and add it to the fixed container. Create a scrolled window and add it to the last created container. Create a text view and add it to the scrolled window. Create a horizontal container box. Create a single line text entry field and add it to the horizontal container box. Create the close stream button and add it to the horizontal container box.

`enter(self, widget, entry, buffer, application, streamName, view)`
The text contained by the text view will be shown in the scrolled window.

`newStream(self, widget, entry, application, dialog)`
The displayed `dialog` box will be destroyed. Call the available method in the `application` to create a stream. Create a fixed container and append it to the notebook. Create a container with two panes arranged vertically and add it to the fixed container. Create a scrolled window and add it to the last created container. Create a text view and add it to the scrolled window. Create a horizontal container box. Create a single line text entry field and add it to the horizontal container box. Create the close stream button and add it to the horizontal container box.

`cursorChangedHandler(self, treeview, buffer)`
Add location, ip and port of the selected element to the text view and show it in the scrolled window.

`receiveList(self, messages)`
Receive the list of streams provided by the application.

`upDate(self, streamName, newData)`
Receive updates provided by the application and add them to the text view. The updates will be displayed in the scrolled window.

`create_list(self, buffer)`
Create a scrolled window. Create an empty list and display it in the scrolled window.

`insert_text(self, buffer, text)`
Add `text` to the text view.

`create_text(self, text)`
Create a scrolled text area to display the specified `text`.

Chapter 6

TESTS

Mobile Chedar Middleware Tests

All the implemented functionalities were tested during the project to identify problems and to assure software reliability. Good performance results were achieved. An official demonstration was also provided. The Figure 26 illustrates a communication between two nodes. The first node starts a query to request a resource. The resource is found in the second node and provided through a resource reply.

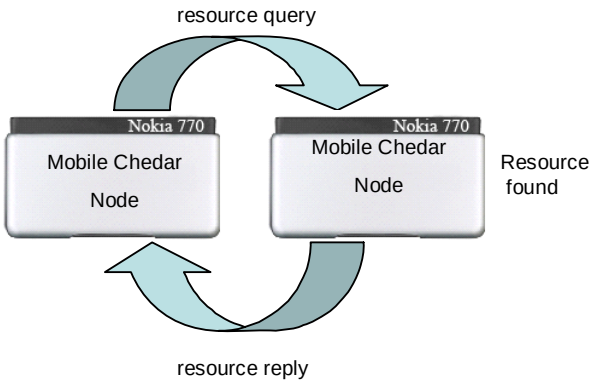

Figure 26 – First test: resource query and resource reply.

The Figure 27 illustrates a communication among three nodes. The first node starts a query to request a resource. The resource is not found in the second node

and the query is forwarded. Finally the resource is found in the third node. A reply is immediately provided by the third node and forwarded by the second node.

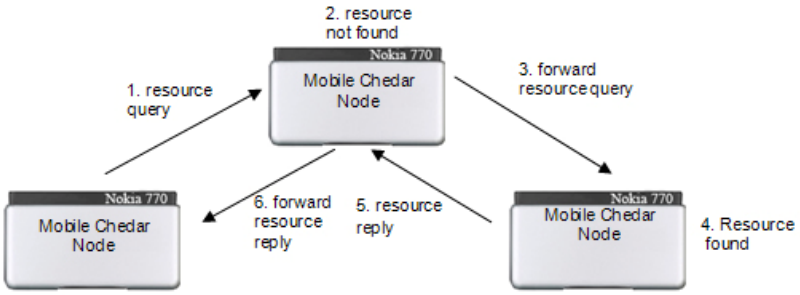

Figure 27 – Second test: resource query, resource reply and forward query.

The Figure 28 illustrates a communication among four nodes. This test assures that the above functionalities also work properly with more nodes. However the main goal was to test the functioning with different values of ttl. When ttl was zero after being decreased, the message was always dropped.

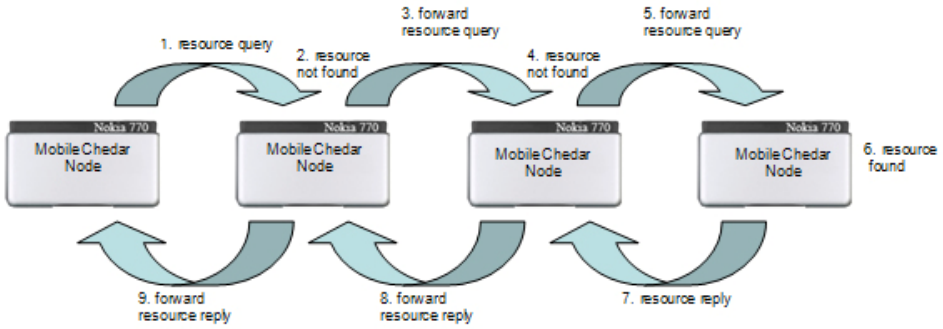

Figure 28 – Third test: resource query, resource reply and forward query.

The Figure 29 illustrates a communication among four nodes. The node located on the left is sending a query to its neighbors. The node located on the right will receive two forwarded messages with the same content. The main goal of this test is verifying if a repeated message is dropped.

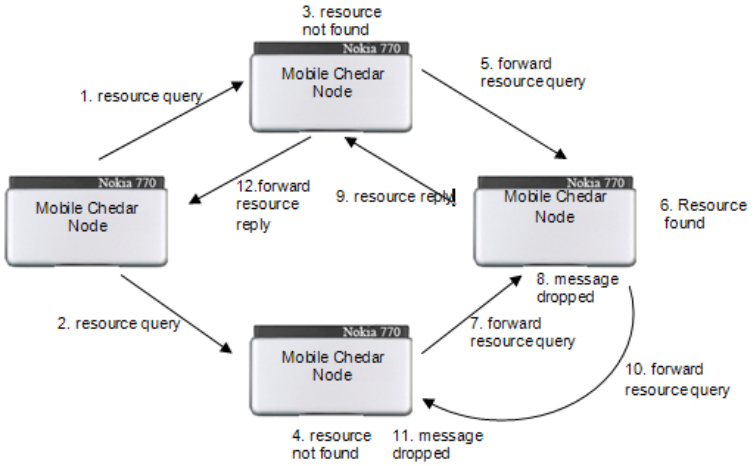

Figure 29 – Fourth test: resource query, resource reply and forward query.

The Figure 30 illustrates a communication among four nodes and verifies the above tests with a different topology.

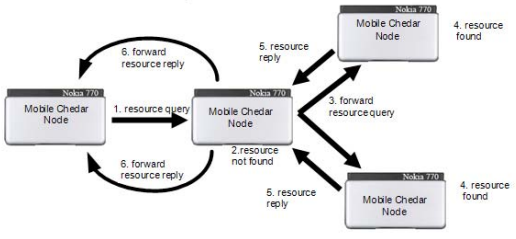

Figure 30 – Fifth test: resource query, resource reply and forward query.

Figure 31 illustrates Mobile Chedar working properly with Chedar P2P Network.

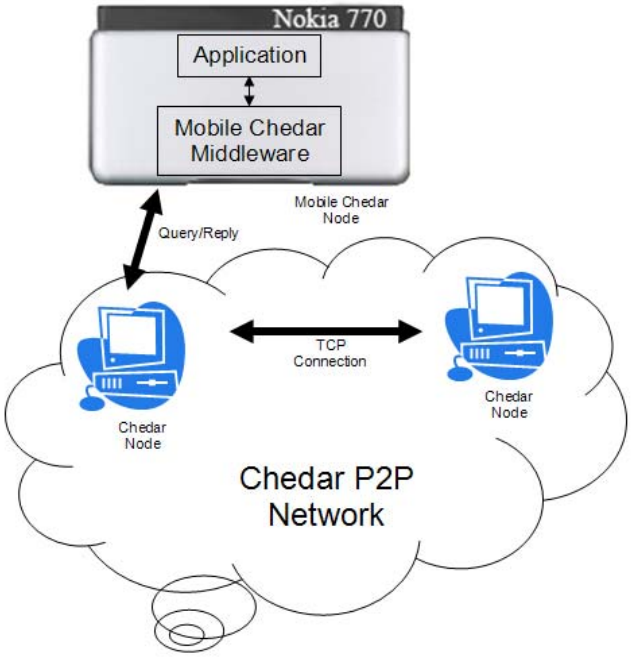

Figure 31 – Sixth test: resource query, resource reply and forward query between Mobile Chedar and Chedar P2P Network.

MP2P Communication Center Tests

Several tests were provided on the application side. The Figure 32 illustrates a communication among four nodes. All the application functionalities worked properly in these tests.

Figure 32 illustrates a communication among four nodes. The main goal of this test was verifying if the updates and joins to the created stream were properly working. The node on the left joins to the stream "Special Assignment". It will send also an update with "Final Report" to its neighbor. Finally the update will be forwarded to all the neighbors that subscribed the stream "SpecialAssignment".

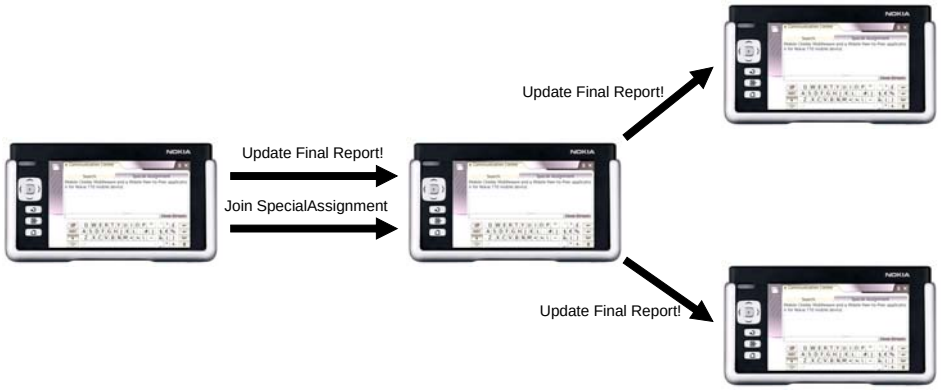

Figure 32 - Join Message, Update messages and forwarded updates.

Chapter 7

FUTURE WORK

Tests

The future work concentrates on field and performance testing. The field tests should be done to determine how the software works when it is used by multiple devices in a lecture. The performance tests should be done to determine the data transfer rates, delays, consumption of processing power and the general suitability to different application domains.

Additional Features

Nokia 770 provides access to the web over WLAN but it is also designed to meet Bluetooth Specification 1.2. Bluetooth communication could be added and used when Wireless LAN is not available.

The neighbors of a node are specified in the configuration file. Another option is to implement broadcast or UDP multicast to discover all the available devices within the same WLAN network.

Other important features are blocking malicious users to avoid their updates and deletion/modification of rows of the stream to allow manually modifying the stream content. The removed rows won't be visible to all the users that are subscribing the same lecture. The idea is keeping the most import information of the stream from the user's point of view.

Chapter 7 – Future Work

New Applications

Peer-to-Peer technologies have received a lot of publicity lately mainly because of Napster and other peer-to-peer systems mostly developed for distributing music and movies in the Internet. An application to share music and movies can be developed to run on the top of the Mobile Chedar Middleware. It can be used for instance by lectures to distribute video and mp3 of their lessons.

The concept of Peer-to-Peer is increasingly applied to a wide range of areas. An application can be developed for instance to share local maps. This could be useful while we are traveling. When we are lost we can download local maps from our neighbors and find our way.

Chapter 8

WORKING HOURS

This project was divided into following phases shown in the Figure 33:

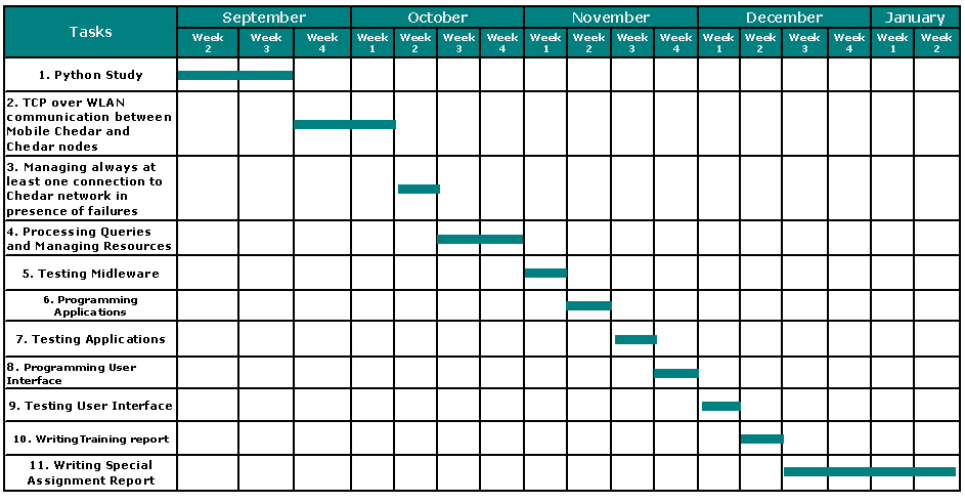

Figure 33 – Phases of the project.

All the stages of the project were undertaken in University of Jyväskylä at Agora Center. In my opinion this was a nice place for working and living with challenge. Before taking this project I never had any experience on research side. This was an excellent opportunity to work with a research team and learn same practices to publish results in articles and scientific papers. The Peer-to-Peer distributed

systems group of Department of MIT is very well disciplined and organized. All needed support was provided during this project.

Chapter 9

CONCLUSION

A Mobile Chedar Middleware and a Mobile Peer-to-Peer application for the Nokia 770 mobile device were developed by me with the purpose of getting an extension of Chedar system to the Nokia 770 mobile device (http://www.nokia.com/770). All established goals were achieved. The Mobile Peer-to-Peer application for group communication in real-time and the user interface were developed as optional features. All the implemented functionalities were tested during the project to identify problems and assure software reliability. Good performance results were achieved. However all the tests were done in a simulator because the Nokia mobile devices are not available yet. The field and performance tests might be done in the real devices to confirm the software reliability. But similar results are also expected from the real devices because Nokia 770 is powered by Maemo with a Linux distribution.

This document contains important information that can be used to write articles and to develop other Mobile Peer-to-Peer applications. It is essential to describe technical knowledge and how to apply it in specific situations.

This was my first practical experience with Mobile Peer-to-Peer Distributed Systems. I learned how to program in Python and use GTK+ on it (PyGTK). PyGTK [5] provides a convenient wrapper for the GTK library for use in Python programs, and takes care of many of the boring details such as managing memory and type casting. After user interface implementation I got a good background on GTK library management [9]. During my degree I have got from my University

(http://www.fct.unl.pt) [10] good technical background. The acquired knowledge was very important for this project. TCP over WLAN was studied in Computer Networks Complements course. A wide experience with several programming languages was useful to achieve a good background with python programming language. The Breadth-First-Search algorithm was studied in Introduction to Artificial Intelligence course. Distributed Systems course gave me a good knowledge about Peer-to-Peer distributed systems. This project was the most important challenge of my studies.

Chapter 10

REFERENCES

[1] Cheese Factory. http://tisu.it.jyu.fi/cheesefactory/index.shtml

[2] Wikipedia.org. http://www.wikipedia.org/

[3] Maemo.org. http://www.maemo.org/

[4] Coulouris G., Dollimore J. and Kindberg T., *Distributed Systems Concepts Design* Addison-Wesley, Third Edition, 2001

[5] PyGTK. http://www.pygtk.org

[6] Kotilainen N., Weber M., Vapa M., Vuori J., "Mobile Chedar – A Peer-to-Peer Middleware for Mobile Devices", Workshops Proceedings of the Third IEEE Conference on Pervasive Computing and Communications (Percom 2005), pp. 86-90, Kauai Island, USA, 2005.

[7] Gmail. http://www.gmail.com/

[8] Google Maps. http://maps.google.com/

[9] GTK Class Reference. http//www.pygtk.org/pygtk2reference/gtk-class-reference.html

[10] Faculty of Science and Technology, New University of Lisbon: http://www.fct.unl.pt/

Chapter 10 – References

For more information see:

- Pilgrim M., *Dive Into Python*. This book lives in http://diveintopython.org/
- Swaroop H., *A Byte of Python*. This book lives in http://byteofpython.info/
- Lutz M. and Ascher D., *Learning Python* O'Reilly, Second Edition, 2003