Proximity Payments with Betala.se

Daniel Henriksson

June 3, 2007
Master's Thesis in Computing Science, 20 credits
Supervisor at CS-UmU: Jerry Eriksson
Examiner: Per Lindström

Umeå University
Department of Computing Science
SE-901 87 UMEÅ
SWEDEN

Abstract

Cash and credit cards are used for millions of transactions every day. These traditional ways of conducting payments have some weaknesses, which are not easily overcome. New payment systems such as Betala.se (developed by Swedish Internet Payments) provide solutions to the shortcomings of traditional payment techniques.

Proximity payments are payments conducted at a short distance between the payer and receiver. This could be for example at a super market or restaurant. The Betala.se payment system was extended in several different ways, in order to be better adjusted for proximity payments.

The first addition was the implementation of a module that can be attached to existing software in cash registers. By attaching this software, the register will be able to receive payments done with a mobile device over Bluetooth.

A custom web application that resembles a checklist was written for use at for example lunch restaurants. This application is particularly useful in situations where it is possible to pay a predefined amount to a receiver, and then have the payment "checked of" a list.

Some Internet surf stations charge the user when visiting certain web pages. These payments are often small and the surf stations are often at remote locations. In order to circumvent the high fees of the credit card system, and the extra maintenance involved in slot machines, a common surf station software was extended to accept payments using the Betala.se payment system.

Contents

List of Figures

Chapter 1

Introduction

Every day, millions of money transactions are made using different kind of methods such as cash or magnetic cards. These methods suffer from weaknesses in terms of security and availability, and therefore alternative payment methods has to be considered.

Proximity payments are payments done between a payer and a receiver at a short distance. The most common methods are cash or magnetic cards, but both of these methods have some disadvantages. Alternative payments methods have the potential of being cheaper (especially for the receivers), quicker, more secure *and* less hardware dependent than for example the credit card system. These methods can use existing hardware such as mobile phones and this means that the costs for hardware can be considerably lowered. A lot of the fees associated with magnetic card are used to cover the losses from frauds and security breaches. With a system designed for security from the start, these costs will not be as high. Compared to using cash, these systems can become superior in terms of easy access, protection against theft and robbery as well as providing an easy overview of sent and received payments.

In this thesis, three different extensions to an existing electronic payment system have been designed and implemented. These extensions have been done as independent sub-projects, but they all have the concept of proximity payments in common. The first sub project is a Bluetooth module written in Java. This module can be added to existing point of sale software systems. By adding this module, customers will be able to perform electronic payments with their mobile phones or other handheld devices. The second project was a modification to SiteKiosk[1], a software running on Internet surf stations. SiteKiosk can be configured to require money in order to let the customer surf the Internet. This software has been extended to accept payments using the new electronic payment system. Finally, the third extension is a web application written for use at for example a restaurant. The web service is designed to resemble a notebook where customers that have payed in advance can be checked in a list.

This thesis has been done in close collaboration with Swedish Internet Payments Ltd. This company has developed a system that supports electronic payments between any company or person. The payment system is called Betala.se[2] and the word *betala* means "pay" in Swedish.

This thesis includes an in-depth study about Bluetooth. Bluetooth is also known as the IEEE 802.15 standard which is the formal name for this technology. The in-depth

[1] www.sitekiosk.com
[2] www.betala.se

study will discuss some basic Bluetooth technology, but the study is focused on how this technology is used from different versions of the Java programming language.

Chapter 2

Proximity Payments

The term *proximity payments* refers to electronic payments that are conducted at a short distance between the payer and the receiver. This could be for example between a pair of mobile phones or between a mobile device and an immobile counterpart. Sample receivers could be a cash-registry at the local store or a vending machine in the company lunchroom.

The most common way of performing proximity payments today is by cash or a magnetic card. Both of these established methods of conducting money transfers have weaknesses that are not easily avoided:

Cash:

- Dealing with cash is a great expense for the society. On a national level, the creation and distribution of currency is expensive and cumbersome. Companies such as stores and banks also spend quite a lot of time and resources dealing with and protecting cash.
- The customer has to get in contact with a bank or go to an ATM[1] in order to acquire cash. Even then, it might not be possible to pay someone the intended amount without having some change.
- Some receivers avoid dealing with cash because of the risk of robbery and theft.

Cards:

- Receivers have to register in order to accept payments with cards. This means that cards are generally restricted to be used in customer - company relations, and can not be used to send money between two individuals.
- Large fees are imposed on someone accepting card payments. These fees exist both as a starting fee upon registration and a smaller fee on every transaction being made.
- It takes at least a day before the receiver gets the paid amount.
- When paying with a card, a malicious receiver has all the information needed to charge the payer several times.

[1] Automated teller machine

– At some point, the receiver has to access the Internet in order to complete the payment.

As an alternative to the above, Swedish Internet Payments has developed mobile phone support to their on-line service. The service is called Betala.se[2] and the system use mobile phone numbers as the account numbers. Money can be deposited to the Internet based account in the same way as a normal bank account. The money can then be transferred to other users either by using an Internet browser or through a small application that runs on a mobile phone. The mobile phone application that Swedish Internet Payments has developed is called MPay. This MIDlet is freely available from the Betala.se website.

Using this technique to perform proximity payments is possible, but would require the receiver to confirm that the transaction has been completed. This can be done using a computer or mobile device to display the transactions through a web interface. Using regular browsers is is a big advantage since the service then is easy accessible without the need of additional software. A disadvantage of using browsers is that using this kind of verification might not be suitable for all receivers.

To be better adjusted to proximity payments, the MPay application was expanded by Swedish Internet Payments. The technology is called BlueCash and will allow the payer and receiver to communicate using Bluetooth. When the receiver has specified the amount, an encrypted payment request is sent to the Betala.se servers. This request is sent using either the Internet connection of the receiver, or GPRS through the mobile phone. The method used depends on the receivers access to the Internet.

The Betala.se electronic payment service has the following properties:

– Payments can be sent to and received by anyone, using the receivers mobile phone number as the account number. If the receiver is not registered in the system, a new account is created.

– The service has low fees. This is especially important for smaller companies that are unable to get favorable deals with the credit card companies. Sending money in this system is always free of charge. The receiver is charged after the first 30 transactions each month.

– Since mobile phones contain a small processor, security can be enhanced by the use of cryptography.

– The sensitive information used for the payment will never be visible to the receiver. This prevents anyone from repeating the transaction.

– By using the Internet access of the mobile phone, it does not matter if the receiver is on line or not.

– The system is dependent on the Internet and sometimes also the mobile phone system. In situations where either the mobile phone network or the Internet can be used the up time of the system is very high.

– Any payment system has to obtain a critical mass of users. This has not yet been accomplished for the Betala.se payment system.

– The system is not integrated with the banks. This means the users need to transfer money between their bank accounts and their Betala.se account.

[2] www.betala.se

Chapter 3

Bluetooth from a Java perspective

3.1 Introduction

Cordless and wireless solutions have seen a major upswing during the last decades. Everything from mice to printers, handheld gaming devices and keyboards slowly started to evolve from a wired solution to a wireless or cordless one. Behind the scenes, the Bluetooth technology is largely responsible for this.

It started back in 1994 when Ericsson started a survey about a low-cost, low-power communication device that could be used by mobile phones to connect to their accessories. Four years later, Ericsson formed a group called the Bluetooth Special Interest Group (commonly abbreviated SIG) together with IBM, Intel, Toshiba and Nokia [5].

Today, the SIG has more than 7000 members including many of the biggest software- and hardware companies around the world. Over a billion Bluetooth devices have been produced, which averages more than 2.5 devices per second since the initial research survey was initiated, back in 1994 [4].

Bluetooth is used to form quick temporary networks between all sorts of devices. This can be done without external media such as cables, and without the need of specifying frequencies or channels. From a user point of view, this technology has made communication quick and easy between mobile devices and computers, for example between a mobile phone and a desktop computer. The developers perspective will be further elaborated in this chapter.

3.2 Technology

Wireless communication is conducted either by light or by radio. Infrared technology for example is based on light waves as means of communication. Bluetooth is a radio technique along side the TV, the radio or the cellphone. These are all regarded as quite established and standard technologies. Bluetooth operates in the 2.4 GHz - 2.483 GHz frequency range which is a free industry frequency band. Referred to as the *Industrial, Scientific and Medical* (ISM) radio bands, these frequencies are free to use for anyone. Because of this, Bluetooth and other technologies such as garage openers, wireless Internet, walkie-talkies or even microwave ovens may interfere with each other.

To cope with the interference problem produced by other devices working on the same frequency band, Bluetooth employs a technique known as Frequency Hopping Spread Spectrum. What this means is that the devices will split the frequency range into 79 small channels. Every second, each device will change channel about 1600 times in a pattern which is agreed upon when the network is initiated. This means that even if other devices are operating on some of the channels, the majority of them can still be used for communication. Bluetooth can dynamically adapt the hopping pattern in order to avoid frequencies used by other ISM devices [5].

The hopping takes place between the transmission or reception of different packets. One packet usually take up one *slot*, but some circumstances may allow for packets to use more than one consecutive slots. One slot is 625 micro seconds and this equals 1/1600 of a second [5].

Bluetooth use an asymmetrical inquiry procedure in order to be discovered by, or discover, nearby devices. Devices that are "discoverable" listen to incoming inquiry requests and send responses. The inquiry procedure takes place at the physical layer, and higher layers in the architecture are not used during this phase. Once two devices are aware of each others presence, the paging (connecting) procedure can be initiated.

The inquiry procedure can last up to 10.24 seconds in noise-free environments. The same data is broadcast 1024 times which will guarantee that all inquired devices are listening to the common frequency, and will be in the inquiry substate at least once during this procedure [10]. If the inquiry is performed in a noisy environment, the time may far exceed the default time of 10.24 seconds and this still does not guarantee that all devices are found [10]. This is because the packet that is sent when both devices are on the same frequency might be corrupt.

The connecting procedure is similar to discovery, but the paging procedure makes use of targeted communication instead of general broadcasts [5].

When two devices are connected, the Service Discovery Protocol provides the functionality to discover what services are offered by the remote device. If the server hosts the desired service, the querying device request a separate connection on which the service is used.

During the connection phase, the devices will determine a master and a slave device. Another device is capable of joining the same network, denoted a *piconet*. The piconet can consist of up to eight devices at the same time, with a maximum of seven slave devices and one master device. A single device may be member in more than one piconet, but may only be master in one piconet at the time. This is because the hopping sequence is determined by the clock and Bluetooth device address of the master device. Having the same sequence in two piconets would render them both useless. Several piconets which have common members will form a network called a *scatternet*. The common members can act as routers for traffic going through the scatternet. Figure 3.1 shows a sample network topology, with one scatternet and two piconets.

In order to reduce the interference created by other devices, the Bluetooth technology has a set of power classes where the most appropriate class can be employed in terms of output power and range. See Table 3.1 which is based data from [6].

Devices are able to dynamically adjust their output signal strength. The target strength is obtained by stepwise adjusting the output power while monitoring a parameter called the Receiver Signal Strength Indicator (RSSI), which is a measurement on how well the target device is able to receive the signal. By lowering the output signal strength of the local device until the RSSI of the remote device starts decreasing, the minimal output power can be obtained.

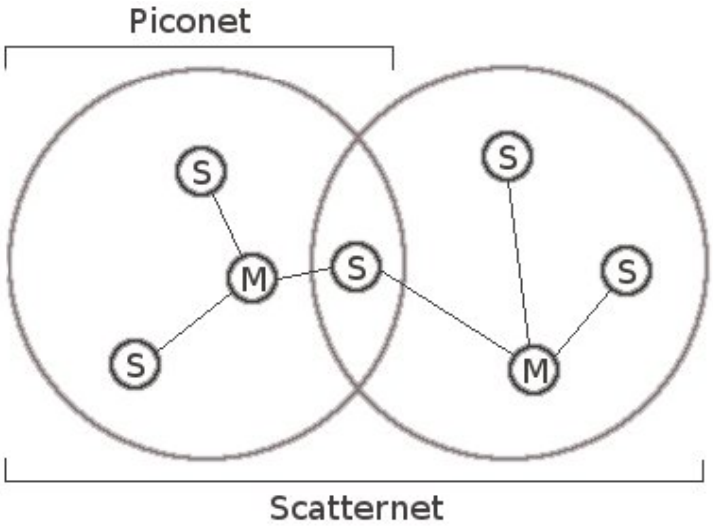

Figure 3.1: Sample Bluetooth network topology. Nodes marked "M" are Master devices, and nodes marked "S" denotes slaves. A piconet is a network of devices all connected to the same master device. Several piconets can form a scatternet by having a common slave device.

3.3 Protocol Stack

Similarly to many other auxiliary devices, the Bluetooth device needs a set of drivers in order to operate correctly. An important part of the drivers is the Bluetooth stack. The stack can be implemented in soft- firm- or hardware and provides control over the local device. It also provides the opportunity to communicate with remote devices [6]. The stack can be subdivided into several smaller components, called *layers* due to the internal structure of the stack.

Figure 3.2 shows a picture of the interesting parts of the Bluetooth stack, which consists of the following layers:

- *Radio Layer* is where the actual radio communication takes place. The frequency hopping and dynamic output power techniques also reside in this layer, which make them totally invisible to above layers.
- *Baseband and Physical Transport* is the Bluetooth equivalent of a physical layer. This layer manages several very important features such as the ad-hoc network

Class	Power (mW)	Range (meters)
1	100	100
2	2.5	20
3	1	10

Table 3.1: Shown here are the different BlueTooth power classes. Different situations requires different devices in terms of range and power consumption.

structure,[1] Time Division Duplex (TTD), physical channels and links and the error correction.

- The*Link Manager Protocols* primary task is to set up and manage the links and authentication. This is done by exchanging packets referred to as Protocol Data Units (PDUs) which are never propagated into higher layers.
- *Host Controller Interface(HCI)* is the layer which is located between the hardware and the software. When using a USB Bluetooth device for example, the HCI is located on both sides of the USB port. The layer deals with the software side on the computer and the actual transport to the peripheral device. The HCI is also responsible of receiving the data inside the Bluetooth dongle[2]. The layer also acts as a software command interface used in order to access functionality located in the hardware.
- The *Logical Link Control and Adaptation Protocol (L2CAP)* provides multiplexing and demultiplexing services to the surrounding layers. The Maximum Transfer Unit (MTU), which sets the maximum size of packets, is relatively small in the Bluetooth architecture and this makes transparent multiplexing services vital to above protocols.
- *Service Discovery Protocol (SDP)* is used to discover what services are available from the Bluetooth host.
- *RFCOMM* is an abstraction layer that make the Bluetooth foundation invisible to higher layer protocols such as TCP, PPP or IP.

The Bluetooth technology also use profiles. A profile specifies options in each layer of the stack that are suitable for the current application. Profiles are used to increase interoperability between products from different manufacturers.

The Serial Port Profile is a generic implementation that is meant to close resemble the regular serial port paradigm. The SPP profile is one of the most generic ones and also the one employed in this project. As you can see in Figure 3.2, the SPP profile is located right above the RFCOMM layer.

[1] Ad-hoc networks refers to temporary networks created 'on the go', which is the basic paradigm in Bluetooth.

[2] External Bluetooth devices are often referred to as Bluetooth dongles.

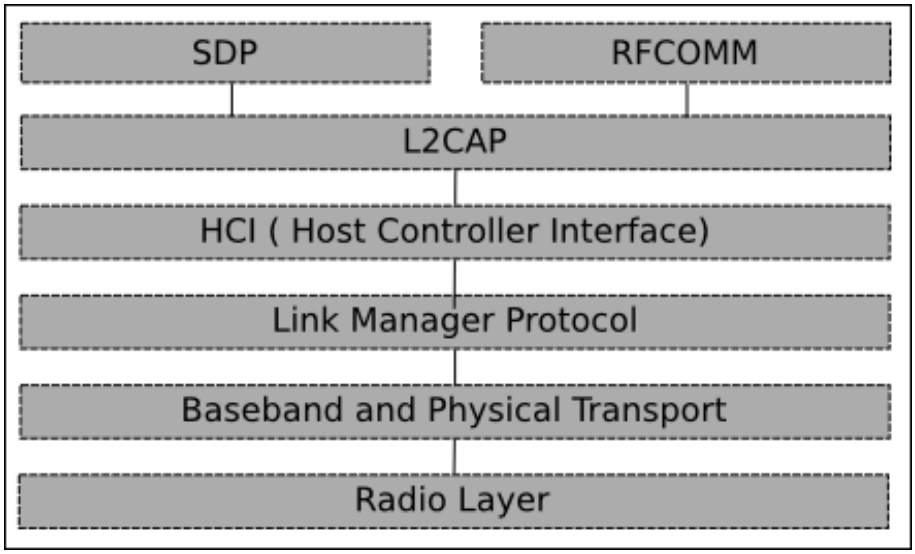

Figure 3.2: The Bluetooth stack. The figure shows the different layers of the Bluetooth stack that are interesting in this context.

3.4 The Java Perspective

The basic paradigm for the Java programming language is to provide a platform independent programming language. This might be the reason why mobile devices supporting applications written in Java is a quickly expanding phenomenon. These devices are limited by the hardware in terms of performance and storage, and because of this a special Java edition has been developed with the sole purpose of running on resource-constrained devices. This smaller version of Java is called Java 2 Micro Edition (J2ME) and the regular one is called Java 2 Standard Edition. Working with Bluetooth on these different platforms can be quite different, despite using Java all along.

There is also a version called Enterprise Edition, which is the standard edition with added functionality that is suitable when producing enterprise applications. This version was not used at all during this project, and so is not further explained.

You might wonder why it is called "Java 2", instead of just "Java". This is because Sun[3] made so many changes between versions 1.1 and 1.2 that they decided to add the "2" in order to properly distinguish between the old and the new versions. The version numbers are kept, which means that "Java 2" starts at version 1.2. To make things even more complicated, Sun did the same with version 1.5, and called the result "Java 5". For some reason, Sun decided that the new names for the different editions should not follow the earlier pattern and be called for example "Java 5 Standard Edition". Instead, they went back to the original notation without numbers. This means that the latest Standard Edition release is called "Java Standard Edition", and the previous one is called "Java 2 standard Edition". See table 3.2 below for a quick overview.

Versions	Title	Editions
before 1.2	Java	-
1.2 - 1.4	Java 2	SE, EE, ME
after 1.4	Java	SE, EE, ME

Table 3.2: Java versions overview. Versions 1.2 up to and including 1.4 are known as Java 2.

3.4.1 Java 2 SE

The Standard Edition packages contain the most commonly used set of classes, without the added functionality of the Enterprise edition. Neither of the three Java versions contains built-in support for Bluetooth, but the way of adding Bluetooth support differs between ME and the other two. In order to use Bluetooth from SE, an external implementation of the Bluetooth stack has to be used. This is because Java SE does not support working directly with the Bluetooth hardware.

There are a couple of different implementations of the Bluetooth stack that can be used from Java, and the most commonly used one are the avetana Bluetooth stack [3] and the Atinav aveLink implementation [2]. The major difference between the two software packages is that avetana is an open source implementation that only supports Linux, while the aveLink version is a commercial product with support for several different operating systems and also different kinds of Bluetooth adapter connections, such as

[3] The company behind Java, www.sun.com

USB[4] and UART[5] based modules.

The economical aspects is probably what determines which stack that is suitable for a particular project. The avetana stack, being open source, is free of charge while a set of five prototype licenses (not for commercial use) for the aveLink implementation is about $4500.

3.4.2 Java 2 ME

Mobile devices usually use a lighter version of Java called the *Micro Edition* (ME). Similarly to the other versions, Java ME does not support Bluetooth out of the box. Due to the restricted software control on mobile devices, adding external Bluetooth software to mobile devices is not as easy as adding it to a regular computer running SE or EE. Therefore, it is up to the vendor of the mobile to handle the implementation of the interface between Java and Bluetooth. This interface is standardized and referred to as the *JSR-82* specification. It would be reasonable to assume that every mobile device which supports both Bluetooth, JSR-82 and Java should be able to run the same MIDlets[6]. Unfortunately, this is not the case. Different mobile manufacturers, or even different models from the same origin, might contain different implementations of JSR-82. This means that each MIDlet has to be tested on each and every mobile device model in order to be completely sure that everything works as intended.

During the development of BlueCash, a whole list of differences between different devices was discovered. These ambiguities arise partly because different manufacturers implement the standard differently, but also because the standard is vague on some parts. If the standard fails to specify a certain attribute, the companies that implement the standard are likely to solve things differently.

To produce a MIDlet that works on most of the mobile phone devices, the code has to be explicitly tested. This is a time consuming task that is very likely to leave some models unverified. Hopefully the future will see more similar platforms for Bluetooth powered Java ME MIDlets.

[4] Universal Serial Bus
[5] Universal Asynchronous Receiver/Transmitter
[6] Java applications running in mobile devices.

Chapter 4

Tina POS BlueCash Plug-in

4.1 Problem Description

Swedish Internet Payments has developed an extension called BlueCash for their MPay MIDlet. The extension is used to make payments between a handheld device and a receiver using Bluetooth. The BlueCash technology use a custom made protocol to allow for a multitude of payment devices and receivers to communicate. This technology use Bluetooth to detect nearby receivers and therefore is better adapted for proximity payments than the standard MPay MIDlet.

BlueCash is well adapted for payments where the amount is predefined, for example when paying to a vending machine. It is also usable for payments at a cafe or super market, but could be better adjusted for that situation. The main reason is that the exact amount is calculated by the receiver, and then entered in the phone by the payer. A better solution is to send a receipt to the payer using Bluetooth. This way both the receiver account number and the amount to pay is already filled in. This reduces the number of steps required in the payment process, and also eliminates the risk of users mistakenly entering the wrong amount or receiver.

4.1.1 Problem Statement

The goal of the project is to implement a working BlueCash Bluetooth module. The module should be better adopted for payments in for example a cafe or store, by sent receipts to the payer where the amount and receiver number is already filled in. The module also has to implement some security mechanisms in order to prevent replay attacks and other kinds of malicious use.

4.1.2 Methods

The BlueCash Bluetooth module project was written in Java 1.4[1]. By using an older version of Java with backwards compatibility, even older computers should be able to run the software. The software was organized as an independent module to make it easy to include the module in different POS software applications.

Working with Bluetooth in Java SE requires an external stack implementation since native Java SE does not support working directly with the hardware. In this particular

[1] Commonly referred to as Java 2

case, the avetana [3] implementation was the primary choice. The main reason is because it is free (under Linux) and thus quite suitable for prototype projects.

The avetana stack is not fully implemented in Java, but relies on native methods in order to access the hardware. This is a drawback since it removes one of the biggest advantages with Java, the platform independence. Even if the actual application does not need to be recompiled in order to run on different system, the avetana stack has to be replaced with the correct platform-dependent build. The stack might have to be recompiled even between different system running the same platform, for example two Linux systems running different versions of the *glibc*[2] library.

4.1.3 Related Work

During his master thesis, Isak Renström created the relevant soft- and hardware needed for the closely related vending machine part of the BlueCash project [9]. The vending machine documentation has been an invaluable source of information during this project. Some parts of the code is also shared between the two projects.

Swedish Internet Payments also developed the mobile phone client while Isak was working on his project. This software has already been tested during the cause of the vending machine development, and so most errors could be traced back to the software involved in the Bluetooth module created in this project.

The BlueCash Protocol was developed by Swedish Internet Payments, and is further explained in the subsection below.

4.1.4 BlueCash Protocol

The BlueCash protocol was developed by Swedish Internet Payments and is used for Bluetooth based proximity payments. The basic concept of the BlueCash protocol is that the receiver and payer should be able to share one of the Internet connections by using BlueTooth to communicate internally. This means only one of the participants in a transaction has to be on line. Many of the modern mobile phones have built in support for GPRS, which is used to surf the Internet with the mobile phone. The BlueCash protocol allows for off line receivers such as a vending machine to tunnel data through the mobile phone, and to the Betala.se servers. The data sent is encrypted using Public Key Infrastructure (PKI) which will protect the integrity of the transaction.

The protocol is request-response based, where requests are known as *Actions* and responses as *Answers*. The messages sent from the client always has to contain the Action parameter. The server will respond to an action command with an Answer message. Both messages have different kinds of required and optional fields, depending on the type of Action. Figure 4.1 shows an incoming message and the message being sent as response. The Action in this case is set to "hello", and the Answer will be of the type "helloResponse". Both messages in this example contain the "version", "language" and "c" parameters. The version is to ensure that the protocol version is the same in both devices. The language parameter in the "hello" message is the language that the client would like to use, and the language parameter in the "helloResponse" message is which language will be used. This means the client can suggest a language, but the server always decides what language is suitable. The "c" parameter is an estimation of the quality of the device current Internet connection and the device with the best access to the Internet will communicate with the Betala.se servers. In the vending machine for

[2]The library working with compiled c programs.

example, the helloResponse message will always have a "c" value of zero. This means that the device is offline, and the Internet connection has to be done using the mobile device.

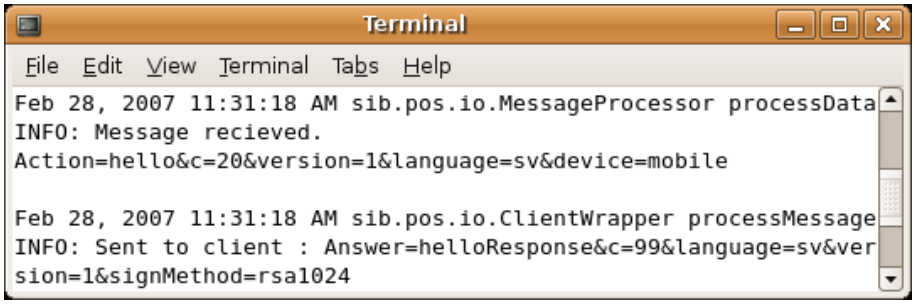

Figure 4.1: BlueCash message exchange. The first message contains an *Action* parameter that shows that this message is a request. The response sent by the server contain a *Answer* parameter. Each message also contains some required and optional attributes.

4.2 Accomplishment

The developing process was roughly divided into the sections outlined below.

4.2.1 Preliminary studies

Since this project was closely related to the previous work of Swedish Internet Payments, studies of the existing software and documentation was a good starting point. General knowledge of how to work with Bluetooth from a Java perspective was also obtained by studying on line examples and the Hopkins and Anthony's book *Bluetooth for Java*[6].

4.2.2 Listing the demands

In order to be able to plan the project properly, a set of demands on the module where defined:

- The module should load specific settings from an external file.
- Security data should be encrypted and stored locally.
- The message processing should strive to be stateless and deterministic. This means the module should produce the same response when given the same input.
- Support for multiple clients should be implemented by creating a new object for each client. The object should contain everything that is specific to the client referenced.
- Internet access should be determined on run-time, and changes has to be dynamically detected and processed.
- Communication to the payment servers should be done through HTTPs.

4.2.3 Choosing the Bluetooth stack

After the initial studies, some communication examples were written to further study how the communication is managed from the Java platform. In order to access the Bluetooth device, external software has to be used and finding suitable software proved to be a bit harder than it seems.

The sample applications that had been studied where either using the aveLink software or the avetana BT stack. The avetana stack was the primary choice because of the open source accessibility and economical aspect. At this stage the project proceeded with the design and implementation before discovering that the stack being used have some drawbacks. A Bluetooth device is technically capable of supporting up to seven concurrent clients, forming a piconet. The avetana BT stack however only supports one client at a time which means that it is possible for clients to connect prematurely to a device, or connect to the wrong device by mistake, and thus blocking the device for the legitimate and intended client.

The early testing done with the half finished BlueCash implementation had also shown that the link between the mobile device and the BlueCash server sometimes remains running despite closing the connection on both ends. This is probably due to a software bug in either the avetana stack or the mobile device Bluetooth implementation. The combination of one client maximum and zombie connections means that the communication sometimes becomes unavailable for incoming clients.

In order to cope with this, the aveLink stack trial version was obtained and the code was slightly adjusted to run with this other stack instead. The aveLink software does support up to 7 concurrent clients, and the zombie connection issue never occurred running this stack. However, the drawback is that the trial version is available for the Windows operating system only and that the trial license expires after 15 days. After this period, a purchased set of licenses is required in order to continue using the stack. A set of five prototype licenses is about $4500, and if the stack is to be used commercially another kind of license has to be obtained.

The final decision was that the cost of the commercial product is above the limits for a prototype project like this, and the drawbacks of the open source implementation has to be accepted as a part of the prototype.

4.2.4 Module layout

The BlueCash module was designed to have several independent parts dealing with different tasks. A main class deals with the interaction between the different parts of the system.

The parts where designed using a few basic directives:

- As much code as possible should be reused from the vending machine part of the project.

- Each part should have no references to other parts of the system, apart from the main class.

From the demands stated above, the internal structure of the module was designed. The structure is described in text below, and also graphically represented in Figure 4.2.

Internal layout:

- The BlueCash class is the main class which keeps references to the other major parts of the system. All input and output data are processed through this class regardless of origin or target.
- A class called Pinger is running in a separate thread. This thread is constantly monitoring the Internet access by attempting HTTP connections to a set of servers defined in the settings file. Changes in the connection state will be notified to the BlueCash object by use of Observer / Observable semantics.
- The code needed for the HTTPs communication with the main payment servers was already created by Swedish Internet Payments. The BlueCash object contains a reference to the relevant object.
- A ClientWrapper object is instanced once for each client that connects to the system. Within is the connection handle, the streams and everything else that the client needs during the session. The BlueCash object keeps a vector of clients currently connected.
- The MessageProcessor is a deterministic and almost completely stateless class that generates the answer message from the server, given a client message as input. This class is also responsible for the security involved in the protocol since different replies and actions are generated depending on the message validation. In order to check the integrity of received messages, the MessageProcessor makes use of the Encrypted class to store and read security data from the disk. This data consists of an anti-replay attack number, which in this case is the latest transaction id. The Encrypted class deals with the encryption and decryption of data and will store the data on disk using the DiskModule class.

4.2.5 Creating the software

Once the layout was planned, the software development process was initiated. The first step was to work out the initialization of the stack and the main loop that the BlueCash module would run. The code needed to create the Bluetooth service was written before carrying on to the MessageProcessor.

A lot of the code for the MessageProcessor part and the relevant subclasses where already present from previous projects. The existing code was written for Java running on a small embedded system so especially the IO part was not quite suitable for use in a Java SE application. Since the core of the code could remain intact, modifying the existing data for this particular project were quite fast and was also the first part to be implemented.

The next step was to implement an early version of the ClientWrapper class to deal with the incoming clients. A small text based graphical interface was constructed in order to test the module out without the interaction with other software. Module specific testing was conducted at this stage to ensure that the existing parts work as intended.

The HTTPs communication code was quickly added along with reading settings from a file. The only major part remaining at this stage was the development of the ClientWrapper class. The class is used to properly maintain the flow of data inside the module.

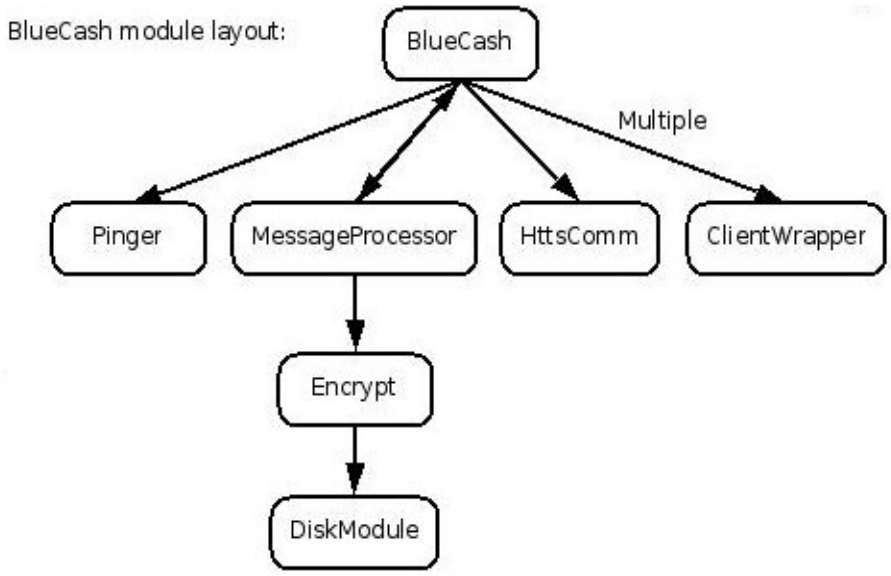

Figure 4.2: BlueCash internal layout. The figure show the different parts of the BlueCash BlueTooth module. Each node in the figure represents a class in the application.

As the module was finished, testing was performed in order to detect and eliminate as many bugs and software glitches as possible. Once the software worked as intended, device tuning and interaction between the BlueCash module and an existing graphical POS system was conducted.

4.2.6 Tuning the Bluetooth device

There are several parameters in the Bluetooth HCI layer that can be altered in order to reduce the time that it takes to find a matching service in the vicinity. The problem is that a lot of these settings has to be set both in the mobile device and in the BlueCash server. The system has to be a general system which any mobile device supporting Bluetooth and Java is able to access, therefore only tweaking at the server side can be done.

On the server side, the *Inquiry Window* and *Inquiry Interval* setting could be tampered with without any adaption in the client unit. The Inquiry Window is the time that the serving device will listen to an incoming connection on a particular frequency. Similarly, the Inquiry Interval is the period of time between two consecutive Inquiry windows. Lowering the Interval means the device will listen to incoming connections more often, and thus increase the chance of finding a querying client. See the Section 4.3 for the small study that was conducted on the different settings.

4.2.7 Connecting BlueCash to Tina

A quite important part of this project was to show that the BlueCash software module could be employed in regular Point of Sale systems. An open source POS software

application was obtained and the BlueCash Bluetooth module was attached. The POS software chosen is called TinaPOS [1]. See the results section below for screen shots of the running application with the BlueCash module attached.

TinaPOS has a list of different payment methods displayed as a vertical tab pane which is accessed once the total sum of the goods has been obtained, see Figure 4.6 in the results section. BlueCash where implemented in TinaPOS by copying and modifying an existing tab, and only a few lines of code where added to other parts of the system. The extra lines where responsible of initiating the BlueCash object on start up, in order to allow clients to access the system before the payment window were accessed.

4.3 Results

Special external software is required to run BlueTooth under Java. A project which is dependant on platform dependant external software will surrender one of the biggest benefits of Java, the platform independance. External software also means extra costs and exposure to software glitches and bugs. The biggest advantage is that the technology is generally available and quite suitable for the short range communication that this project makes use of. The quality of the implementation on the mobile devices differs a lot which soon became apparent during the project.

There are some Bluetooth parameters that where customized to obtained faster device and service discovery. A small study was conducted to try out the different settings on several different mobile phone devices. Each device was also tested in two different environments. In the first scenario only a few other could be found. In the second case about eight other devices where discovered in the vicinity. The tests where conducted using a separate MIDlet developed by Swedish Internet Payments. The MIDlet has the sole purpose of running a set of tests and one of the tests is measuring the time required to find BlueTooth devices. By running this test and measuring the time to find the same Bluetooth device each time, test data was collected. The SonyEricsson phones available (k610i model) where unable to run these tests. This is because the discovery process implemented does not provide the discovered devices dynamically. The phones instead perform a device discovery for ten seconds, and then report the set of devices found.

Since the Bluetooth device discovery process is a bit unreliable, and therefore the data collected contains huge variations. This is expected, but the variations make the data obtained hard to analyze. Using the same settings and device the time to discover the device varied with about factor ten. With different settings on a device, the greatest span shows a factor 85 variation between the highest and the lowest value obtained. The two devices where attached to a table to keep the distance between the mobile unit and the receiver constant. In total, 70 different values where obtained and compared. With the variation taken into consideration, it seems that a inquiry interval of 512 slots reduced the time needed in order to discover the target device. This value is one forth of the standard value (2048 slots). The Inquiry Window setting did not seem to have any obvious effect on the results.

The BlueCash module uses the Java Logger functionality to keep a log of recent events. When running BlueCash with the built-in text based interface (and through TinaPOS for that matter) the log strings will be printed to the console. Figure 4.3 shows the output when initiating BlueCash including the initialization of the avetana stack. A warning is printed if the server runs a key which is not meant for production machines. The figure also reveals that the machine running the program has a working

Internet access. The information is stored in the log together with the time stamp when the connection was discovered.

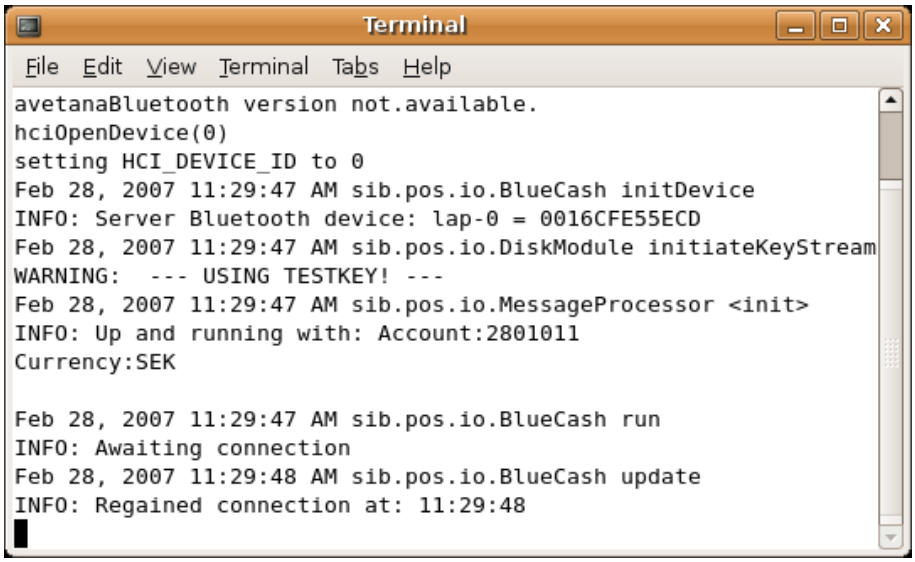

Figure 4.3: BlueCash initialization output. The first few lines shows the BlueTooth stack initialization output. The BlueCash module will show the local device address along with a warning if a test key is used. Shown in the figure is also the output produced when the BlueCash module is waiting for an incoming client along with a message informing that the Internet connection has been regained.

During the course of the application, the Pinger class constantly monitors the Internet access with a configurable frequency. The monitoring is done by attempting HTTP connections to a set of predefined servers. The Pinger will detect a change in the connection and log the results. Figure 4.4 shows a scenario where the Internet access is momentarily lost and subsequently regained a couple of seconds later. The host in this case is "www.google.se".

If the Internet connection of the receiver goes down, the connection measurement value described back in the BlueCash protocol section (4.1.4) will be automatically updated. This allows the systems to start using the Internet connection of the mobile device instead. The total up time of the system therefore can be better than the up time of the receivers Internet connection.

The graphical interaction with TinaPOS was done after the module had been sufficiently tested using the simple text interface. TinaPOS, with the basic interface shown in Figure 4.5, was expanded with the BlueCash module in order to allow payments from mobile devices.

The modification to the graphical POS software was mostly conducted in the tabbed pane where the different kinds of payment methods are selected. Figure 4.6 shows the pane which is used for regular cash. This was subsequently expanded and modified to provide the additional functionality. The button panel on the right side is used to enter the sum payed by the customer, in order to calculate the change.

Figure 4.7 shows the modified panel used for BlueCash. The button labeled "Charge"

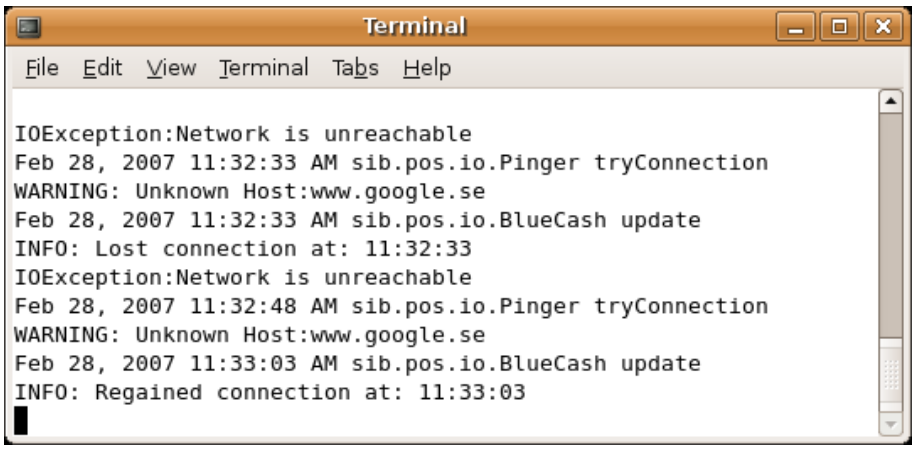

Figure 4.4: Pinger class output. The output shows the Pinger class detecting a problem with the Internet connection. When the problem is detected, the BlueCash module will be notified and the message "Lost connection" is then printed by the BlueCash module. The Internet connection is regained after 30 seconds in this example.

is used to send the total sum to the BlueCash module. The sum is passed along to the client for confirmation. The lower button, "Disconnect client" is used as a work-around to the problem introduced by the software stack. This solves the problem with premature or inproper clients blocking the access for the intended customer.

4.4 Conclusions

BlueCash is able to provide a proximity payment method using Bluetooth and Swedish Internet Payments' technologies. The module provides an easy to use interface for any Java based POS system, and requires no additional hardware apart from a Bluetooth device.

The software stack currently employed has some drawbacks, but these drawbacks are not severe enough to justify the cost of commercial products. The limitations enforced by the stack includes the limit of one concurrent client and the problem with zombie connections. The problems have been solved by adding the option of disconnecting any active clients. When the system is used commercially, the investment in another software stack is motivated. When still dealing with prototypes, the cost of such software licenses are too high to make them a reasonable alternative. Another solution would be to implement an interface between Java and for example the generally available BlueZ stack implementation [7].

The use of Bluetooth as a communication technology has both advantages and disadvantages. The pros include the great availability in mobile devices, the cheap hardware and the ease of extending regular computers to support Bluetooth devices. The major drawback is the greatly varying implementations in the mobile devices. This forces the software to be customized to cope with the differences present.

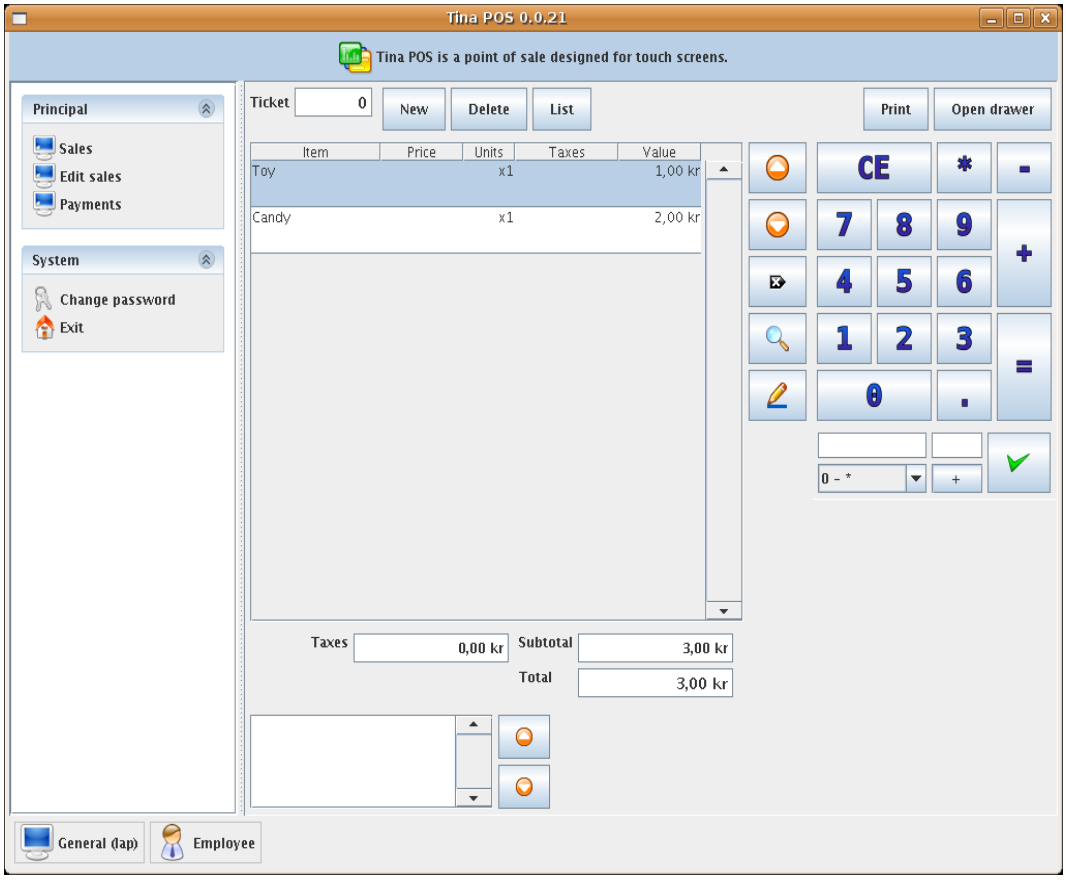

Figure 4.5: Tina POS software. This figure shows the standard view of the Tina POS software application. Items are shown in a list in the center of the screen.

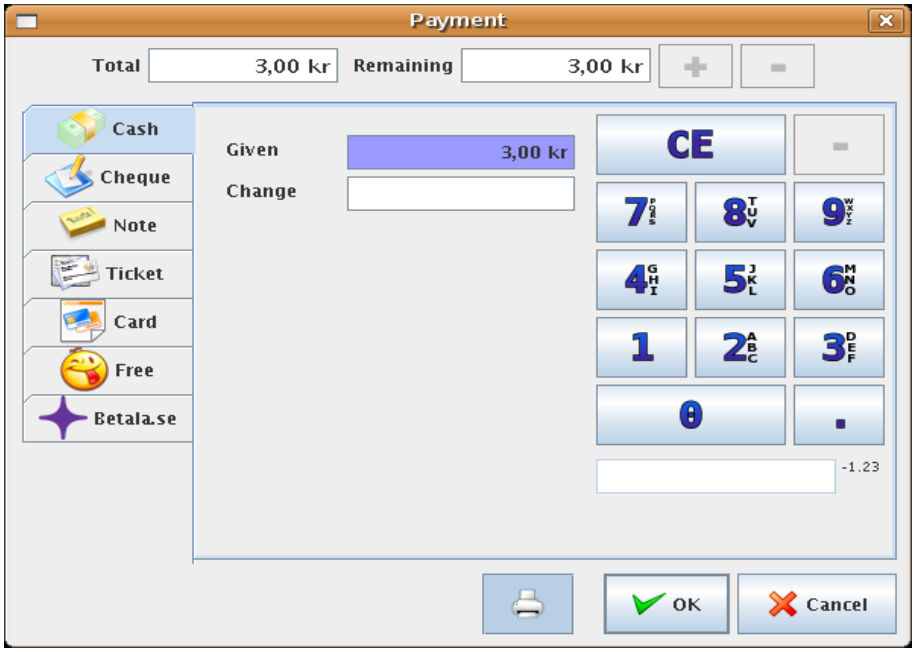

Figure 4.6: Tab used for payments with regular cash. The number pad on the right is used to enter the given amount and calculate the change.

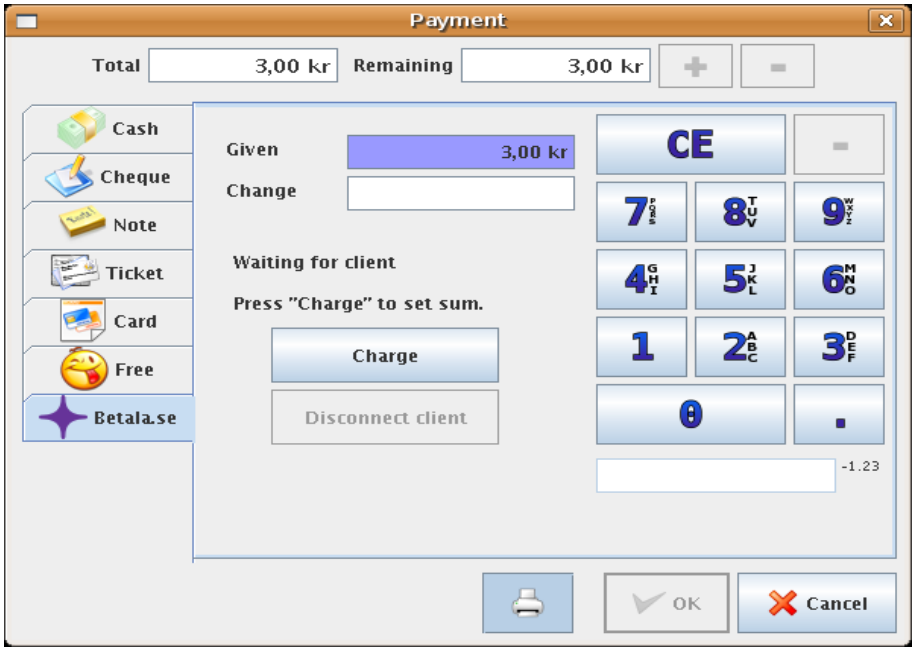

Figure 4.7: The tab from 4.6 was copied and extended with the functionality required by BlueCash. The button labeled "Charge" is used to send the total sum to the BlueCash module. Another button labeled "Disconnect Client" can be used to force a disconnect on the current client.

Chapter 5

SiteKiosk payment device

5.1 Problem Description

Public surf stations are getting more and more popular. Tourist centers and resorts alike have caught the trend of offering Internet access to their visitors on a restricted basis. Stations offer a free surf zone for pages relevant to the host of the station, but charge the use of other pages. For example, a tourist center might allow their visitors to check out the local tourist pages and local community website, but charge the use of sites like Hotmail[1] or Google[2].

The current ways of conducting these proximity payments are by cash, using a slot machine, or by the use of credit cards. These payments suffer from the same drawbacks in this situation as they normally do. Using coins, the surf station has to be physically monitored. Someone has to take care of the money once the container gets full, and also to ensure that no one smash the box open and steal the money. For credit cards, the high fees per transfer make it inprofitable for the station owner. This is beacuse a lot of people make small transactions, and the fee is roughly the same as for larger transactions.

The Betala.se payment service is quite optimal for this kind of transactions. The user does not need anything physical such as a card or cash. As long as the user remembers his mobile phone number, and the login credentials, the non-free surfing can be paid using Betala.se. By adding the Betala.se website to the list of payment free zones, the user is able to administrate their account from the Internet station before paying. Since the service is fully electronic, no extra hardware such as a slot machine or card reader is required at the station.

Many stations use a software application called SiteKiosk[3]. SiteKiosk is a Windows application that can be automatically started when the computer is booted. SiteKiosk will override some parts of the Windows operating system, taking control over security related procedures. The application can be configured to limit what processes that can run on the system, the hardware access and several other parts of the system. This is in order to prevent unintentional use. The SiteKiosk application is developed by Provisio[4]

By extending this software through the implementation of a Betala.se payment device

[1] www.hotmail.com
[2] www.google.se
[3] www.sitekiosk.com
[4] www.provisio.com

plug in, users at surf stations will be able to use Betala.se when paying for their surf time.

5.1.1 Problem Statement

The purpose of this sub-project was to implement a Betala.se payment device plug in for SiteKiosk. This will enable the use of the Betala.se payment system at thousands of Internet stations around the globe.

5.1.2 Methods

SiteKiosk has a native support for payment device plug-ins in two different ways. The first alternative is to create a DLL[5] using a SDK[6] provided by Provisio. The second is to use a build in scripting device, which provides a JavaScript interface between the plug in and SiteKiosk. For the Betala.se plug in, the scripting device will be employed since this is closer reassembling the current system and also is a more generic solution.

For the graphical part, SiteKiosk already contains a lot of the graphics needed and these will be adapted to suit the needs of this particular plug in.

5.1.3 Related Work

The SiteKiosk plug-in described in this chapter is based on existing code already present in SiteKiosk. The application package contains a sample implementation called DummyDevice. This device is meant to be used as a sample device plug-in implementation.

The applet use some existing code from Swedish Internet Payments in order to communicate with the database.

5.2 Accomplishment

The SiteKiosk Betala.se sub-project can be divided into two major parts. The JavaScript code that is executed by the SiteKiosk ScriptDevice, and the Java applet employed to communicate with the database. Netscapes JavaScript and Microsofts JScript are two implementations of the same technology. The term *JavaScript* in this thesis refers to code that can be run on any of them.

The JavaScript code is responsible for the graphical user interface and the user interaction. The code will pass the user input along to the applet. The applet will then contact the Betala.se servers in order to validate the input and return the result back to the JavaScript code.

This was initially intended to be done in JavaScript only. In order to create a HTTPs connection to the Betala.se servers, the actual JavaScript to be signed with a certificate. SiteKiosk features a custom Internet Explorer (IE) implementation as the browser, and therefore requires signed Active X controls. Mozilla Firefox, which is the second largest browser [8], has a totally different signing process. Code that is signed for one of the browsers can not be run on the other.

Supporting more than one kind of browser is not that important for this particular project since the SiteKiosk browser is IE based. Limiting the code to a single browser

[5] Dynamic Link Library, Windows shared libraries.
[6] Software Development Kit, some sample code to make the development of new elements easier.

makes it harder to reuse the code. Therefore, creating browser dependent web application should be avoided when possible.

5.3 Results

SiteKiosk accepts several different kinds of payments. The different alternatives are configured for each Internet station by using the SiteKiosk configuration utility. Once the device has been added, it will appear in the list of payment alternatives shown in Figure 5.1.

Figure 5.1: Shown are two different Sitekiosk payment alternatives. By choosing the Betala.se alternative, the user can be charged for the surf time from their Betala.se account.

The SiteKiosk package contains existing graphics which are used in these dialogs as well. Using the same set of graphics makes the plug-in link visually to the other parts of the SiteKiosk application.

Choosing Betala.se as the desired payment alternative will generate a dialog where the transaction parameters are specified. In this dialog, credentials for the user account are specified along with the sum to pay. The GUI will also show the resulting surf time for the amount specified. The whole dialog is shown in Figure 5.2.

Once the sum has been configured, the data is sent to the applet and a process bar is displayed to the user while the applet is communicating with the Betala.se servers. Refer to Figure 5.3 for a picture of the validating dialog window.

The transaction will either be approved (Figure 5.4), or rejected (Figure 5.5) by the

Figure 5.2: In the user details dialog, the account information and the amount is specified. The surf time obtained by paying the specified amount will also be shown.

Betala.se servers. If the transaction was approved, SiteKiosk will be notified by the JavaScript code and the user will be able to surf for the specified amount of time.

5.4 Conclusions

The Betala.se SiteKiosk payment device plug-in was implemented using both JavaScript and an applet. Signing the JavaScript would probably have been a more elegant solution, but the limitation imposed by the two concurrent signing standards would mean that a single solution would only run on a single type of browser.

Using the applet as the engine of the device means that Java have to be installed on the Internet stations which are to use the Betala.se payment service. This is a limitation since ideally, the number of running processes on such stations should be kept low.

Taking the DLL route would probably have been too much work for this thesis. This is because no code from earlier projects would have been reusable. All applications produced and maintained by Swedish Internet Payments are based on Java. By implementing this device in C++, the maintenance of this particular code would require extra knowledge and effort from the permanent staff.

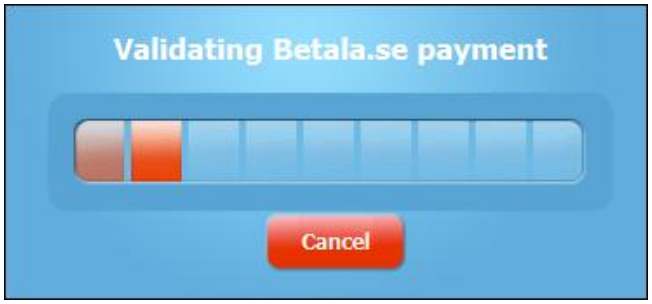

Figure 5.3: This figure shows the dialog displayed when the payment is validated.

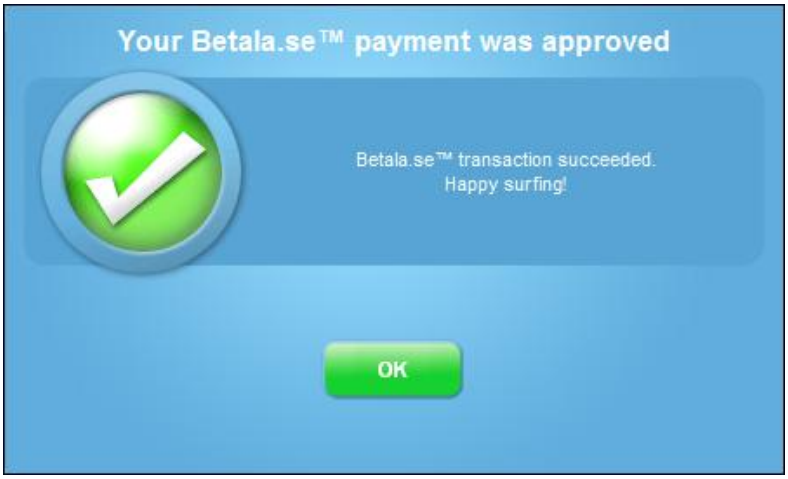

Figure 5.4: If the payment is approved, the above dialog will be displayed to the user.

Figure 5.5: This dialog shows the sample output of a failed transaction.

Chapter 6

Check Project

6.1 Problem Description

The Betala.se is a technically mature system that has been running stable for more than six months. The system is still in an expanding phase in terms of users, so by providing Betala.se payment solutions to nearby companies the number of users in the system will hopefully grow. The goal of this sub-project is to allow for a local restaurant to accept proximity payments using the Betala.se payment system.

The restaurant use an embedded point of sale system with restricted possibilities of modifying the POS software. This means the interaction can not be done using the BlueCash Bluetooth module system. Therefore, alternatives has to be found and developed in order to connect the restaurant to the Betala.se payment system.

Using an external small device as an alternative to a full fledged browser seems to be a good solution. The choice felt on a Palm TX handheld because it has a large screen and built-in support for both Bluetooth and Wireless Internet.

In order to make the system communicate with the handheld device, a custom Betala.se web service has to be implemented. The service should be designed to be used on a handheld device, but still have to be accessible from any regular computer.

6.1.1 Problem Statement

The purpose of this project is to connect the local restaurant to the Betala.se payment system. This should be done by implementing a custom *servlet*. A servlet is a Java application that generates a HTTP response dynamically when receiving a HTTP request. The servlet is to be accessed through the PDAs native browser.

6.1.2 Methods

Swedish Internet Payments current web applications are all written in *Java Server Pages* (JSP). By following the same layout as the other applications, the maintenance will be more similar to the already existing services.

The Palm TX device does not have to be altered by adding any software. This is because all calculations are done at the server, and the browser only needs support for HTML in order to function properly.

6.1.3 Related Work

Swedish Internet Payments already employs servlets using the same internal structure as the one created in this project. Some of the code from previous servlets has been re-used in this project as well, but the biggest similarity lies in the structure and layout of the application.

6.2 Accomplishment

The servlet was implemented using Java Server Pages (JSP). The servlet has a central controller class that generate the next page to load. The controller is queried with a page parameter and then returns a reference to the suitable page. The controller class will receive an Action parameter which shows the controller what action was performed. The controller class also have access to the "HttpResponse" and "HttpRequest" objects where outgoing and incoming data can be accessed.

For example, the link "www.betala.se/check/index.jsp?action=login" would result in the controller class receiving the parameter action "login". The controller will use the data stored in the request object to silently validate the user data.The resulting JSP page depends on the success of the validation. Figure 6.1 shows a graph of which pages can be accessed depending on the current page. In addition to the listed ones, inproper arguments to the controller, such as an invalid Transaction ID, will generate an error page.

The servlet is divided into two separate processes. One is the controller structure which handles session specific queries, and the other is a servlet used to access the database. The database servlet class is called WebApp and a single instance of this class is run on the server, regardless of the number of sessions. The controller class can access this object through the context class. Running the common methods in a singleton manner will reduce the workload of the server.

The WebApp class provides the following functionality:

- Maintains the connection to the database and provides methods for the most common tasks.
- Contains and calculates the upper and lower time limit for transactions, since only daily transactions are to be shown.
- Provides encryption and decryption functionality that is used by the cookies.

This service is to be used on a restaurant with a PDA[1] completely dedicated to this task. The servlet therefore was designed to be used with a PDA browser. During the log in sequence, only read only access is permitted in order to raise the level of security and prevent malicious use. As a consequence, it is possible to store the whole log-in information in a cookie on the handheld device without creating a severe security flaw. With the log in information already stored, only two clicks are required in order to access the page of incoming transactions. Keeping this process quick and simple is very important since reduced delays when performing proximity payments is one of the major keys of this thesis.

The user is able remove the cookie from the main menu, if the device is used to monitor more than one account.

[1] Personal Digital Assistant, a common name for handheld devices.

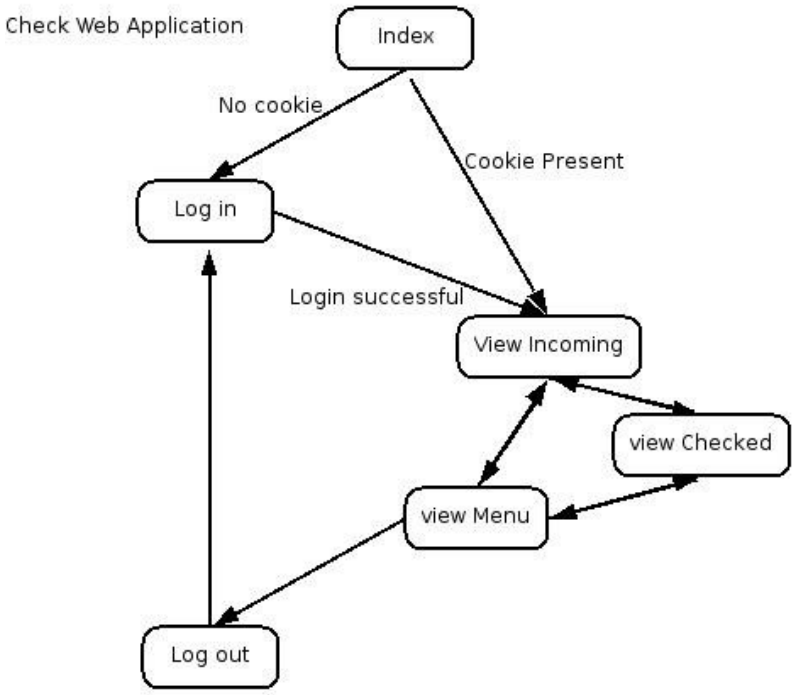

Figure 6.1: Check web application page overview. The arrows shows which target pages that are accessible from any given source page. Some accesses (marked with text) are conditional.

6.3 Results

The first time the page is visited, the log-in page will be displayed. This page is shown in Figure 6.2. By entering the required credentials, the user will either be taken inside the service or sent back to the log in page. A check box in the lower part of the page will allow the user to store his or her credentials. If the data is stored, the log-in page will be bypassed on the next visit.

If the phone number and PIN provided are valid, the user will be taken to the Incoming page, where incoming transactions are listed. Figure 6.3 shows this page.

The web service has been designed to resemble a classical checklist. This analogy will help the user to understand the graphical interface, and also helps the user to understand what this service does.

When the customer has arrived at the restaurant, the employee can use the check box by the customers name to mark that transaction as used. Using this kind of checking system serves two purposes; The first is that only people which has not yet arrived are shown in the list of incoming elements. This makes it a lot easier to find the listing in question. Second, this also prevents two customers from stating the same name and thus cheating the system. Figure 6.4 shows the updated view which is generated by checking one of the customers in the list.

Figure 6.2: Shown here is a Palm TX screen shot of the login page. The phone number and read only PIN code has to be entered in order to gain access to the system. A checkbox allows for the log in data to be saved as a cookie.

Best case scenario is that the employee never have to go beyond the list of incoming transactions. However, if something goes wrong or if the employee wants to confirm that a customer have made his or her payment, the servlet also provides a list of previously checked customers. This view is shown in Figure 6.5.

More detailed information is available by clicking the name of the customer. This will generate a page with more detailed information for the transaction. Figure 6.6 shows this view. If the details page is accessed through the view of checked transactions, the employee also has the option of moving the transaction back to the list of unchecked transaction. This operation is deliberately done in two steps, in order to prevent unintentional use.

The "Menu" link can be clicked in order to access the main menu of the application. The menu contains some information about the current account such as the current balance. As seen in Figure 6.7, it is also possible to terminate the session and log out. This can be done either by clicking the "Log out" link, which will terminate the session, or by using the "Log out and delete cookie" alternative, which will prevent the user from being automatically logged in at the start of the next session.

Once logged out, the user is presented with the page shown in Figure 6.8. Here, the user is presented with the alternative of log in in to a different account regardless if the cookie from the previous session has been deleted or not.

6.4 Conclusions

Working with Java Server Pages in this environment has proved a pleasant experience. By moving the complexity inside Java and keeping the HTML[2] pages fairly simple, the task of creating this service was greatly simplified.

This sub-project had a smaller workload than the two previous ones, but provided

[2]Hyper Text Markup Language

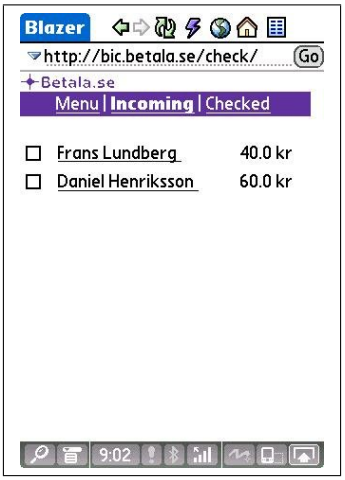

Figure 6.3: Two incoming transactions are shown in this figure. The transactions can be "checked" by clicking the checkbox on the left. By clicking the name of the payer, details of the transaction can be obtained. The details screen is shown in 6.6.

some new and interesting challenges. Being able to use Java for the more complex situations made the process easier since the complexity of the actual pages where kept to a minimum.

Figure 6.4: On this screen shot, one of the transactions shown in Figure 6.3 have been checked. The transaction is then moved to the list of checked transaction (Figure 6.5) and will no longer be displayed as incoming.

Figure 6.5: This page shows a list of "checked" transactions. The details of the transaction can be viewed by clicking the name of the payer (see Figure 6.6). By viewing the details, the transaction can also be moved back to the list of incoming transactions.

Figure 6.6: This figure shows a view of the details page. Additional information such as the message can be viewed from this page. If the details page was generated from the list of checked transactions (as in this case), the user also has the option of moving the transaction back to the list of incoming transactions.

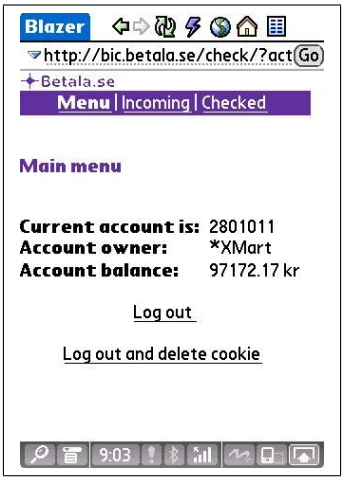

Figure 6.7: Shown here is the main menu where some account information is displayed along with two different log out options. By clicking either of the log out links, the user will be taken to the log out page (Figure 6.8).

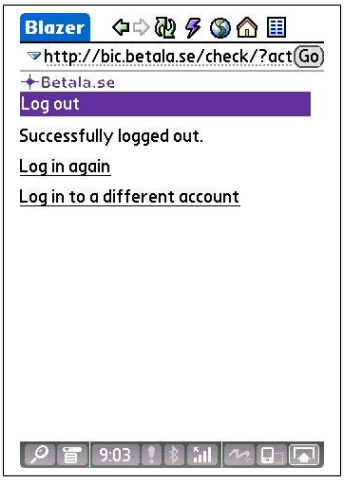

Figure 6.8: The log out page contains two links, and both will take the user to the log in page (Figure 6.2). The difference between the two alternatives is that the "Log in again" option will use the cookie (if any) to log in immediately. The other alternative "Log in to a different account" will bypass the cookie and force a manual log in.

Chapter 7

Thesis conclusions

Proximity payments are one of those things we do every day without reflecting about them. New technologies for these kind of transactions are emerging, and Swedish Internet Payment's system Betala.se is one of them. Perhaps, in a few decades, ATM queues, money counting and armoured car robberies will be things of the past.

The Bluetooth module written for Java based POS systems is one major step forward for the Betala.se payment system. With major advantages compared to cash or magnetic cards in terms of reduced costs, availability and security, this technology really has the potential of becoming globally employed. The implementation of this software was less time consuming than expected because of a detailed protocol specification, some existing code from similar projects and a well tested client. The result is a very interesting piece of software with an enormous potential.

In some situations, extending the POS software with the Java based Bluetooth module is not possible. Some POS systems use software that is very closely connected to the hardware and others might lack Bluetooth support or USB ports where an external Bluetooth device can be attached. For these situations, the Check web application is a great alternative. By using external hardware the application can be used regardless of the existing systems. Since the actual payments are done in advance, the time spent on each customer drops significantly and less time is spent for the customer waiting in the queue. The end result is satisfying, even though the areas of use for this particular extension is slimmer than the Bluetooth module.

One of the biggest assets with the Betala.se payment system is the ability to conduct small and spontaneous transactions. This could for example be purchasing candy from a wending machine without heavy fees or having spare change. Another example is paying for surf time at public surf stations. With the Betala.se payment device plug in written for SiteKiosk the payments can become easier and more profitable for the station owner. Developing this extension was an interesting task with some new challenges compared to the previous ones. The language, JavaScript, for example was a totally new experience. Implementing something against a given interface in an unknown language means a lot of reading and studying examples. The result is a nice addition to the existing SiteKiosk software that can be easily globally distributed.

I believe that the expanding functionality of alternative payment methods, such as Betala.se, will considerably reduce the card and cash dominance. The advantages of the new technology are significant enough to make an impact on the more traditional ways of conducting payments.

Chapter 8

Acknowledgments

I would like to thank Swedish Internet Payments for the opportunity to do this thesis in their regime. They have provided me with massive support both during the development process of the software and the writing of this document. My university supervisor Jerry Eriksson has given me invaluable feedback and constructive criticism especially on the report.

References

[1] Adrián Romero Corchado. Tina pos - home -. Web site, March 28, 2007. http://tinapos.sourceforge.net/, date visited given.

[2] aveLink. avelink: Bluetooth software and technology solutions. Web site, March 28, 2007. http://www.avelink.com/, date visited given.

[3] avetana. avetana jsr-82 implementation. Web site, March 28, 2007. http://www.avetana-gmbh.de/avetana-gmbh/produkte/jsr82.eng.xml, date visited given.

[4] Bluetooth SIG. The official Bluetooth membership site. Web site, March 28, 2007. http://www.bluetooth.org, date visited given.

[5] Bluetooth SIG, Inc. Bluetooth.com | the official bluetooth technology info site. Web site, March 28, 2007. http://www.bluetooth.com/, date visited given.

[6] Bruce Hopkins and Ranjith Antony. *Bluetooth for Java*. Apress, 2003.

[7] Open Source. Bluez - official linux bluetooth protocol stack. Web site, March 28, 2007. http://www.bluez.org/, date visited given.

[8] Refsnes Data. Browser information. Web site, March 28, 2007. http://www.w3schools.com/browsers/, date visited given.

[9] Isak Renström. Bluetooth payments to non-connected devices. Masterthesis, Umeå Universitet, 2007.

[10] Ryan Woodings, Derek Joos, Trevor Clifton, and Charles D. Knutson. Rapid heterogeneous connection establishment: Accelerating bluetooth inquiry using irda. In *Proceedings of the Third Annual IEEE Wireless Communications and Networking Conference (WCNC '02)*, 2002.