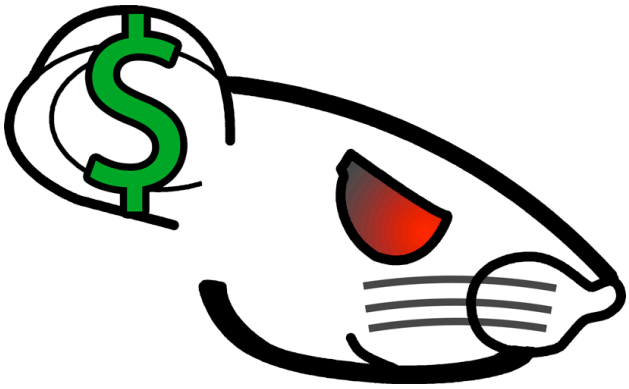


Pricerat.com

Sensor based systems
Final Report

Pricerat.com Karsten Becker, Zhijia Wang, Yuqin Xu

Project goals

Introduction

The goal of this project is to help our customers to save money. We created a service that provides a customer with information about the real value of an item, and we help him to organize his shopping tours.

Scenarios

In our first scenario a person we call Homer is entering a store. He wants to buy an iPod. The price tag reads 300$. He is wondering if the iPod is really that expensive. A short beep coming from this barcode reader and the pricerat.com client tells him not just that 300$ is a total rip off, but also that there is a store around the corner that sells it for 250$.

The second scenario helps people which share a household. While Homer is shopping for an iPod his son Bart found his pleasure on the Duff beer. He drinks it and throws the empty bottle in the trash. Luckily enough the trash-can is equipped with a RFID reader. This registers the bottle and automatically puts it on the family shared shopping list. When Homer sees that the beer is empty he runs to the Kwiki-Mart to buy new Duff beer. He is heading home to enjoy his cold beer in front of the TV. His wife Marge was shopping at the same time. And she also saw that the beer was empty. But she also saw that Homer already bought a new one because he checked it from the shopping list.

Realization

To realize these scenarios 3 components were identified:

- Server component (Karsten)
    - Stores and provides information
- Mobile client (Zhijia)
    - Allows the user to interact with the information
- Smart trash-can (Yuqin)
    - Automatically fills the shopping list for disposed items

Concepts

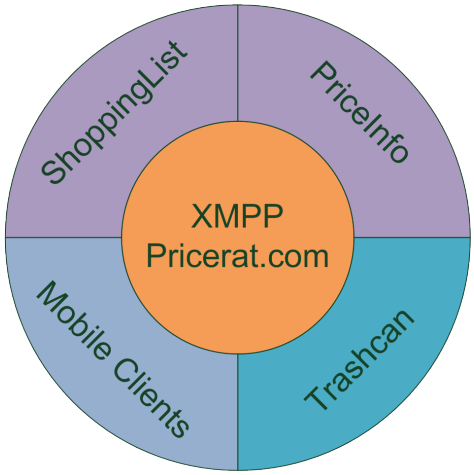
The first concept is the shared shopping list. It allows the user to put items into a list that is centrally stored on a server. This list can be shared with other users. If a change to the shopping list is done in any way it is pushed directly to the subscribed users of that list.

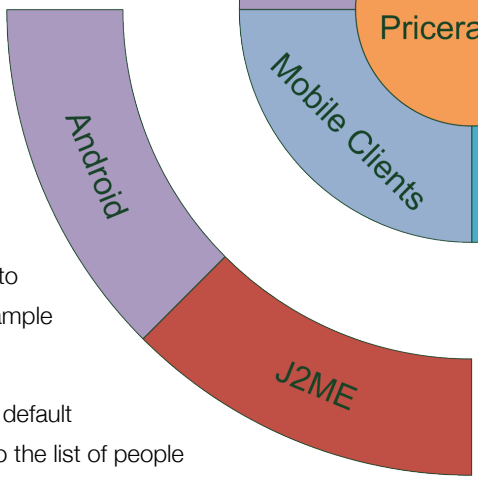
The second concept is the location aware price information. A mobile device that is aware of its location can send a request to the server to ask for cheaper stores that are proximate to the current users location.

Server component

Introduction

The server component has to deal with mobile clients that can not always be connected to the server. It also needs to possibly handle thousands of users at the same time that communicate with each other.





Instead of reinventing the wheel a XMPP server was a natural choice. This handles all the communication features. The openfire XMPP server can handle +60.000 users per IP address with 20% load in an example config.

The information providers are connected as regular users to the XMPP Server.

ShoppingList

A user can send ADD, REMOVE and UPDATE requests to this service. The protocol used is XML. See the XML example on the right side.

This will add an item with the ean 153532525235 to the default shopping list. It also adds 5 DVDs to it. Yuqin is added to the list of people that share this shopping list. And also a new shopping list is created which is called electronics.

The item with the id 5 is removed and yuqin is removed from the list of people that have access to this shopping list. Also the shopping list with the id 6 is removed.

At the end an update is send of all shopping lists.

The interesting thing about the EAN request is that the response may be delayed. In order to resolve the EAN several registries are queried. This is a time consuming process. But the user will receive an immediate response. Once the EAN could be resolved a new update is pushed to the users. This ensures that the user gets a fast feedback of his action and makes the implementation of the clients very simple.

```
<add>
     <item ean="153532525235"/>
     <item amount="5" name="DVDs"/>
     <share jid="yuqin"/>
     <list name="Electronics"/>
</add>
<remove>
     <item id="5">
     <share jid="yuqin"/>
     <list id="6">
</remove>
<update/>
```

ShoppingInfo

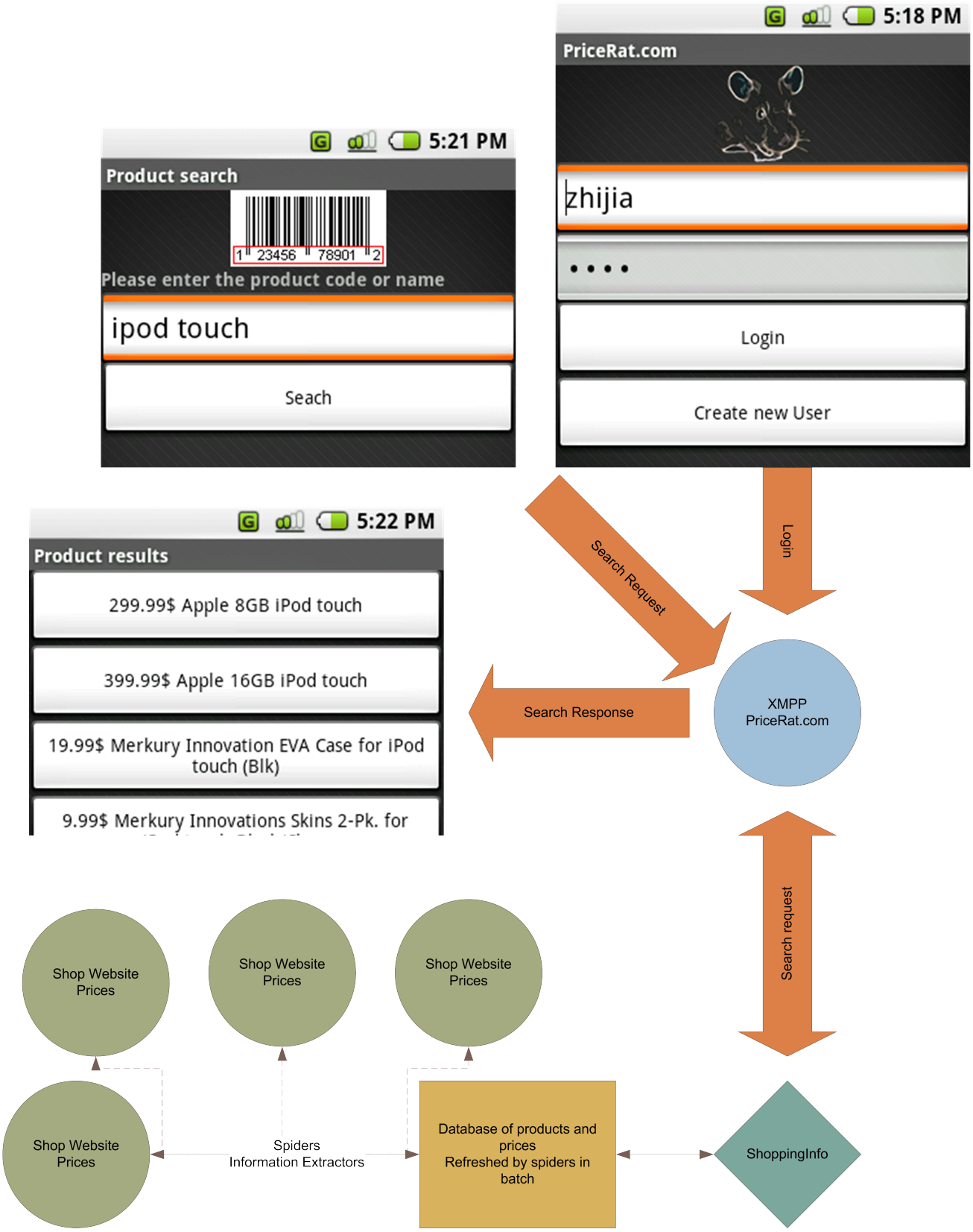
A client sends the name or the EAN (not yet implemented) to the server. The server will responds with a list of items that match the send search string. Now the client can choose to rephrase its search, or pick a result. The chosen result is send to the server together with the current location. Then the server will find the stores that are within a 10 miles radius and return this informations.

EANInfo

This component is not available for end users. It accepts XML queries for an EAN and contacts several web services (currently only GS1). As this might take some time the result is returned when available or an error is reported.

Example communication flow

At first the client logs into the XMPP server. Then he enters the product name. A search request is send to the ShoppingInfo user on the XMPP server. The ShoppingInfo queries it's database and returns the result to the Client which presents it to the user. The database for the products and prices is batch updated during night using spiders and information extractors.



Mobile clients

Introduction

The Mobile client is a software running on the customers'cell phones, which mainly handles the user interface of this service, and through the Mobile phone, users can use the shopping system. The functionality is separated into two aspects:

- User side. Make shopping list, manage shopping list, and use shopping list
- Server side. Share shopping list, update shopping list, and get price and shop location information.

Realization

The prerequisite for this project is a cell phone which can connect to internet. To make things easy, we have chosen a system that has many users and more information. So the Nokia N95 is selected as primary which build on a dominate system—Symbian 60 and has all the function we need, GPS, WIFI, Accelerometer, Camera, and it's also a good choice for later development.

With the latest Software Development Kit from Nokia-- S60 3rd Edition, Feature Pack 2, and NetBeans6.0 as Java development environment, everything for develop a software has been prepared.

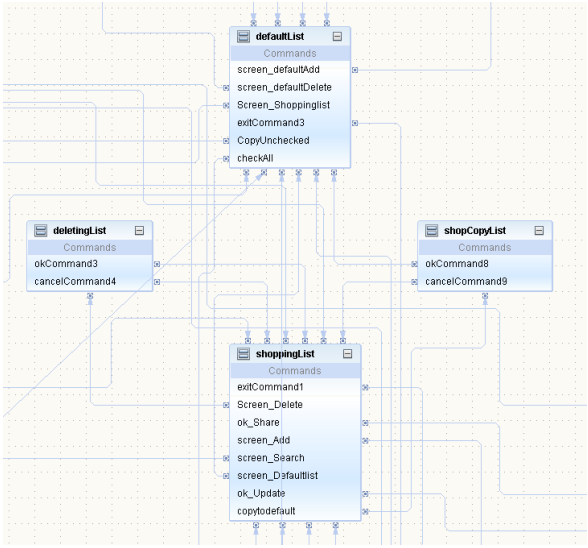
This software system structure is defined in the following parts:

- Default List. It's a user default shopping list, that doesn't need access to Internet. The user can Add, Delete shopping items offline, and copy the items to the real shopping list while online, and it can be stored after the software exits.
- Shopping List. It's a list stored on the XMPP server. The user can access it by Update the list to a phone and Share the list after modification, and copy unchecked items to the default list for offline usage.
- Login. This is used for sending authority to server to ask for getting access.
- Communication. The communication API for communicating between the mobile client and XMPP server
- Alert. Alerting system for all the possible exception, error, failed problem and confirmation success.

- Search Part. This part is used for searching items, get price and location information. (Not finished)

Implementation

There is a Visual designer in Netbeans IDE and an emulator for simulation, which make things easy for me. What I have done is, put those components on the canvas, design buttons on each visual screen, connect buttons and screens by the program flow with wires, write method for each button, control the flow and modify the communication API to be compatible.



Result

The codes have already run on the emulator without any bug by now.

The final step is to generate Jar and Jad file, and install them in N95.

Discussion

There are too much stuffs need to be added, like finish the search part, use the GPS function, decoration for the UI by SVG image and accelerometer. If there is more time, I think all above are available, and I will keep working on these unfinished part.

There are some experiences for IDE: Netbeans6.0 and Eclipse3.3.2; Sun wireless SDK2.5.2 and Nokia SDK 3rd, FP1.

- Netbeans is suitable for entrance level visual design, because when the system is growing bigger and bigger, Netbeans crashes easily when debug and modify visual components. Extremely slow.
- Eclipse is easy to use for a beginner to operate, faster, and NEVER crashed. But the debug function is not as handy as Netbean.
- Sun SDK is easy, but more advance and fast. Noika SDK always crashes and slow, but can emulate the real phone UI.

Conclusion

It's not easy to develop a real software for a new guy first into this Java world, but finally I made it. I learned really a lot from it, from the basic concept of Java, to how to write a Java codes; from how to use an IDE, to configure the IDE for purpose; from how to mimic an example, to write a code as what I need; from how to debug a long code to join each part to a whole program.

Smart trash-can

Introduction

The purpose of this project is to realize the main function of Smart trash can, which contains two aspects:

- Detect and read the tag of the trash in a certain range of area, which we set the range as 1 meter.
- Realize the communication between RI-6 and server.

As the RFID technology becomes more and more popular in applying in industry and its comprehensive application are widely accepted, the idea of using RFID technology to realize the functionality of smart trash-can is very efficient and robust. For this project, we applied the LR-6 WiseMan RFID Reader which is made by TagMaster company with high performance. For instance, with built-in processing and database verification capabilities along with turn-key application software, the Wiseman is ready for immediate, stand-alone operation.[1] with the strong performance and easy use of LR-6, our smart trash-can realize the detection function and totally satisfy our needs and wants.

Realization

On the other part, Smart trash-can should realization of communication between RFID reader and Pricerat.com server. For the development environment and tools we use, we apply Jabber technology with gloox library for C++ programming in Linux server which provides an easy and reliable communication solution.

Because the smart trashcan system is disassemble into two main parts, so we will discuss them with two parts.

Detection part

In detection part, we would like to introduce RFID technology then describe the way I realize the functionality.

Radio Frequency Identification (RFID)

Radio Frequency Identification (RFID) is a method of auto identification that is suitable for identifying both products and assets within the supply chain environment [2]. For instance, RFID has been getting a ton of coverage in the press ever since Wal-mart stated that they were going to require their suppliers to tag products. However, RFID's applications are huge, including our project using RFID in an innovative way.

RFID technology comprises 3 basic elements:

- RFID tag
- RFID reader/writer
- host system

The RFID tag

The RFID 'tag' is essentially a memory device with a means of revealing and communicating its memory contents, when prompted (scanned) to do so. [3] It consists of a piece of integrated circuitry, some memory and an RF antenna. There mainly two kinds of RFID tag. One is active tag, means the tags have a power source within them, such as a small battery. The other is passive tag with a very low power integrated circuit that is able to gain enough energy from the scanner/reader RF signal to actually power itself for long enough to transmit the contents of its memory. With self-supplying power, active

one transmit their information for a much farther distance than the passive tags. The key point of using RFID is simply a matter of reading the associated tag to get the unique ID of the tag. Once you have the unique ID, we can identify the item by the data information. For our project, we apply the semi-active tag which is made by TagMaster Inc.

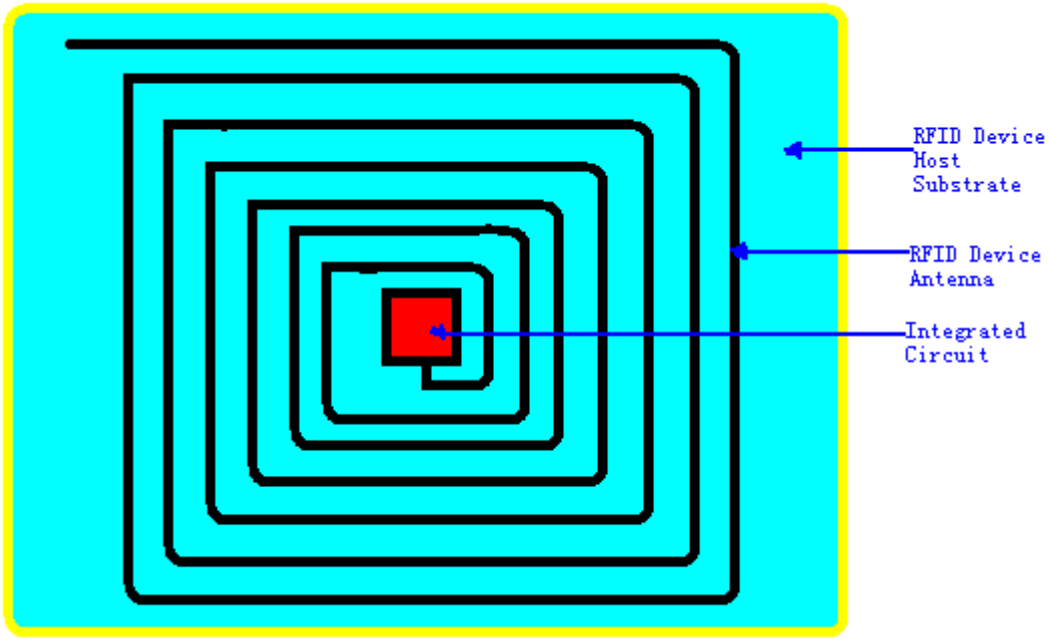


Figure 1. The typical RFID tag principle

The RFID reader

The RFID reader/writer consists of some circuitry and an antenna. In the case of passive tags, the RF field created from the antenna both energizes the tags and picks up their RF transmission of data. In the case of active tags, the RF field reads the tags and may also be used to activate the tag (i.e. switch it on for a programmable period of time). [2]

The host system

The host system is normally a line of business software application. In our project, the host system consists of Linux server and the server of pricerat.com

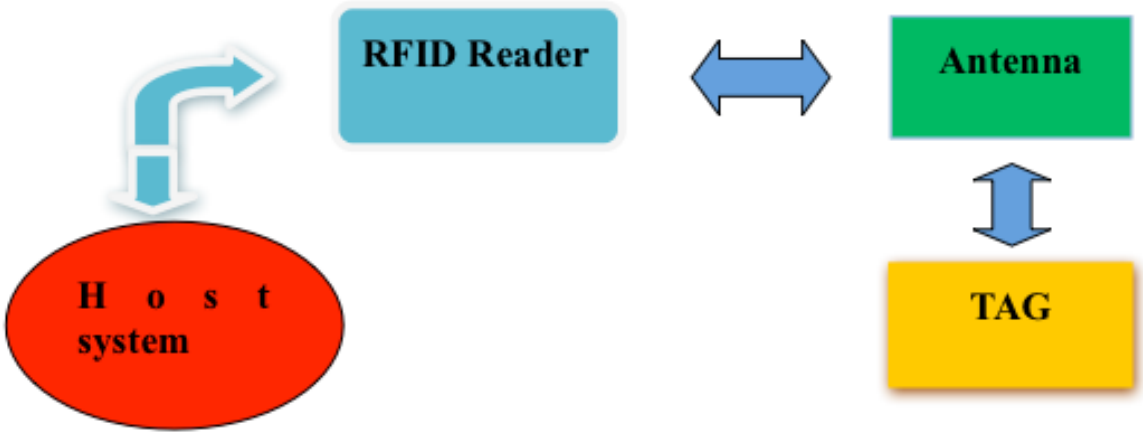


Figure 2. The whole smart trashcan system with RFID technology

Implementation

LR-6 is a programmable, 2.45GHz Read/Write unit for Automatic Identification and verification of stationary or moving vehicles, objects and people. With built-in processing and database verification capabilities along with turn-key application software, the Wiseman is ready for immediate, stand-alone operation. The LR-6 supports several standard interfaces including Ethernet (TCP/IP), RS232, RS485 and Wiegand/Mag-stripe. The reader is designed for future expansion requirements and includes built-in USB Host interfaces, memory card slot and an expansion board interface. [1] With this strong performance and characteristic, detection functionality can be easily implemented. Following are the process:

Connect LR-6 RFID device to Linux server and router, turn on the device by plugging it in. All device setting and network connection have been done by teacher before we start our project.

Set the programming environment. Before I start to program the control and functionality code for the device, I have to set PATH for cross compilation. The step is following: add to badge PATH environment variable `/opt/tm-sdk/gcc-3.4.4-glibc-2.2.5/arm-softfloat-linux-gnu/bin` path to the tagmaster toolchain sot that you can compile and link code. Then making a working subdirectory for compile application for the LR-6. Copy `readid.c`, `Makefile` and `taglib.h` files from `/opt/tm-sdk/readid` directory to working subdirectory.

Programming in VIM development environment in Badge server in Linux sever with the basic readid.c code. By understanding the basic application code and reading the menu of RL-6, I have starting to revise and add new function we need. For instance, `tag_reader_setreadrange()`, `tag_reader_settagspeed(TAG *reader, bool highspeed);` `tag_reader_setreadbeep(TAD *reader, bool state)`. By using these functions, I realize the detection range for 1 meter; sound notification when trash is throwing into the trash can; print out the trade name by translating the tag number into a readable message. Following it's the system initial code:

```
/* create TAGHUB and TAG reader handle with hostname=NULL and port=0 to use default:
localhost/192.168.2.88 and 9999 */

  tag_hub_init(&hub);

//tag_reader_init(hub, &reader, NULL, "192.168.1.61", 0);

  tag_reader_init(hub, &reader, NULL, NULL, 0);
```

Other main function like:

```
// Set the frequency of the output carrier as a "channel" 0 - 99.

tag_reader_setchannel(reader, 77);
```

// Setup the tag filter mode and timeout. If set to off, the timeout argument is not used.

```
tag_reader_settagfilter(reader, TAGFILTER_PERIODIC, 1000);
```

Communication Part

For a robust and intelligent sensor based system, the capability to communicate with other systems or devices which belong to context awareness system is very important. In our project, we provide a very robust and simple GNU solution which based on Jabber/XMPP technology using gloox library for development by C++ language.

There are mainly three function parts to be realized separately:

- Implementation of connecting to the pricerat.com server;
- Sending a message to pricerat.com server to notify system shopping-list updating when RFID reader has detected tag.
- Send notification message once for every new item thrown into the smart trash-can.

Implementation

XMPP (Extensible Messaging and Presence Protocol) is a protocol based on Extensible Markup Language (XML) and intended for instant messaging (IM) and online presence detection. Regarding to the superiority of the protocol that allow Internet users to send instant messages to anyone else on the Internet, regardless of differences in operating systems and browsers, we can easily realize the communication between Linux servers. On the other hand, regarding to cross compiling by using GCC compiler, we choose gloox library which is a rock-solid, full-featured Jabber/XMPP client library, written in C++. It makes writing spec-compliant clients easy and allows for hassle-free integration of Jabber/XMPP functionality into existing applications. The gloox library provides all the class and function we need so that we just need to choose the suitable one and apply in our code. First of all, a class named `Bot` is declared, and it inherit from `Client` and `MessageSession` class. Then we define: `public: Client* j; MessageSession* m_session;` which derived from the gloox class library. Then initialize the `Client* j` like this:

```
j = new Client( "trashcan", "trash","88.198.32.210","trashcan",5222 );
//define the username, password, server ID address of j Client,
//"5222" is the port of XMPP server
j->setSASLMechanisms(SaslMechPlain );
j->registerMessageHandler( this );
j->registerConnectionListener(this);
j->setPresence( PresenceAvailable, 9 );
j->connect();
```

If the initialization is correct, this code will implement the connection to pricerat.com.

Then varieties of member function are declared in following. `virtual void handleMessage( Stanza* stanza, MessageSession* session = 0 )` will implement the function of notifying about incoming messages. `virtual void handleMessageSession( MessageSession *m_session )` will implement the function of notifying about incoming messages by means of automatically created `MessageSession`s. We implement the sending message function by defined as `void send(const std::string & message, const std:: string & subject="")`. This is the preferred way to send a message from a `MessageSession`.

In order to monitor the connection state of system and quickly put right when some connection error happened, some useful functions like `virtual void onConnect(),virtual void onDisconnect( ConnectionError e )`, `virtual bool onTLSConnect( const CertInfo& info )` are constructed in class Bot. `void onConnect()` notifies about successful connections; `virtual void onDisconnect()` notifies about disconnection and its reason; `virtual bool onTLSConnect()` is called when the connection was TLS/SSL secured, for example when the connection meets the Transport Layer Security (TLS) problem, this function will return a value, true if cert credentials are accepted, or else false is returned and the connection is terminated.

Because the RFID device will be periodically running and output the tag number to the server, if every output sends to the pricerat.com shopping-list database which will update automatically every time when it receive message from trash-can, the whole system will be out of control and meaningless. Therefore, we must realize the function that send notification message once for every new item thrown into the smart trash-can. The implementation method is following: first, define an integer array named `tmp[128]`, then using a loop to query whether the output tag number has been saved to the array, if the tag is an element of the array, the tag number will not be sent, or else, the trash-can will send a message to the shopping-list.

Cross compile

When we implement both control part and communication part, we have to start to integrate this two separate file together. Both connection and sending message functions are constructed as class Bot's member function, and we realize a query method to send message to the shoppinglist in `main()` function. In Linux server, we use GCC compile which is a set of

compilers produced for various programming language by the GNU Project. One detail or mistake that we neglect and puzzle for a long time is that if you are including a C header file that isn't provided by the system, you may need to wrap the `#include` line in an `extern "C" { /*...*/ }` construct. After figure out this problem, we use the command:

```
./gloox/libtool --mode=link gcc -Wno-deprecated -Wall messagetest12.cc -o messagetest -I/home/badge1/xuhx1983/gloox/src/ /home/badge1/xuhx1983/gloox/src/libgloox.la -I /opt/tm-sdk/include/ -ltag -L/opt/tm-sdk/lib/host/ /opt/tm-sdk/lib/host/libtag.a
```

This command realizes all the connection of the index and library files of RFID device and gloox. After executing this command, an executable file we named `messagetest` is generated. For some unknown reason, RFID reader could not connect to the internet directly. Therefore, we have to run the code in Linux server by the command `./executable file.`

Discussion

In the final test, all the functions implement quite well. In figure 3 shows the simple model of Smart Trash-can. The RFID reader is on the bottom so that we can get an optimizing detection range. The red line covered area is the actual detection area. Though we set the detection range is 1 meter in code, vertical direction signal is more sensitive than the horizontal direction and the actual detection range is less than 1 meter because both the design of antenna of RFID reader and environment noise take effect to the detection area. In communication test, we use Miranda IM(instant message) software which support jabber/XMPP protocol. Because trash-can are a client in pricerat.com system, when it sends a message to the shoppinglist client, the shoppinglist update automatically and send a notification that the shoppinglist has been updated.

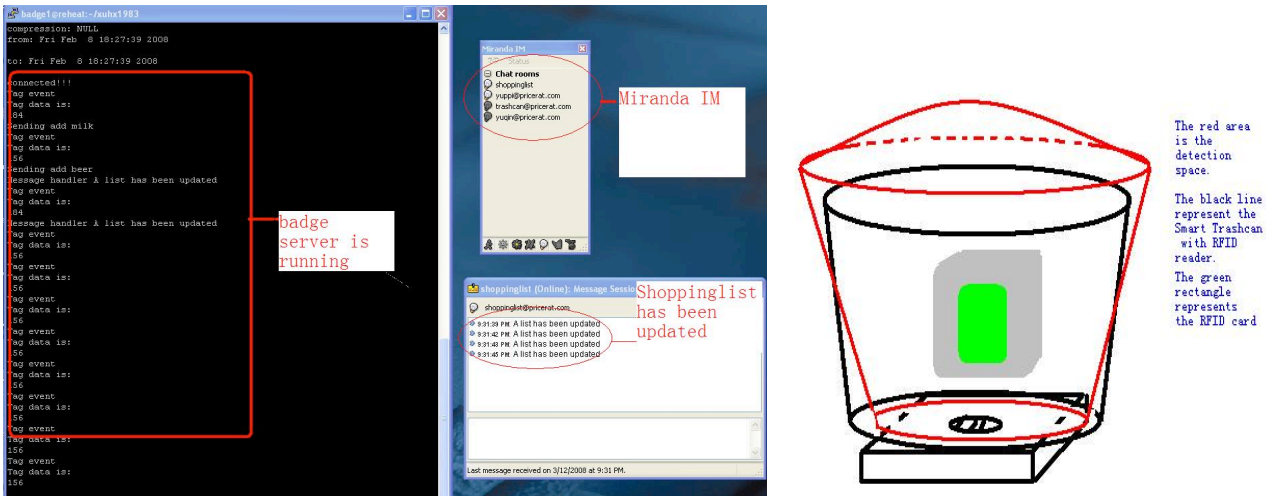


Figure 3.the simple model of Smart Trash-can and the picture of verification of communication result.

Conclusion

Smart trash-can system belongs to a new kind of context awareness system with detection and communication function which has a bright future application especially when it connects to a network server and supports a network service. With the high performance and easy use of WiseMan LR-6 RFID device, we can easily implement the control and detection function. However, in practice, we have to provide a reasonable price of Smart Trash-can for customer, therefore, a cheaper but reliable RFID device will be a better choose regard to the MIPs/Watt/$$ rule.

Reference

TagMaster Inc., TagMaster product sheet, part no.154600, RL-6 WiseMan. www.tagmasterna.com.

Microlise, Farrington Way, Eastwood, Nottingham, NG16 3AG, RFID Tagging Technology, opusworld@microlise.com, www.microlise.com, 3rd January 2003

Pricerat.com

The Institution of Electrical Engineers (the IEE), Radio Frequency Identification Device Technology, July 2005.

Mark. T. Smith, getting started with the tagMaster RFID readers.

http://searchdomino.techtarget.com/sDefinition/0,,sid4_gci935535,00.html.

http://camaya.net/api/gloox/classgloox_1_1MessageHandler.html#_details.

http://camaya.net/api/gloox/classgloox_1_1MessageSession.html.

http://camaya.net/api/gloox/classgloox_1_1ConnectionListener.html.

http://camaya.net/api/gloox/classgloox_1_1Client.html.

http://camaya.net/glooxdoc.

http://en.wikipedia.org/wiki/GNU_Compiler_Collection.

http://www.parashift.com/c++-faq-lite/mixing-c-and-cpp.html.

Readid.c

Taglib.h