# The HomeSIP Project: house automation with SIP

Salim ELLOUZE
Damien LIGOT
Julien MARCHET
Hanitra RATRIMO

ellouze@enseirb.fr, ligot@enseirb.fr,
marchet@enseirb.fr, ratrimo@enseirb.fr

**Abstract**. This project aimed at developing embedded software on a PDA in order to communicate with a temperature sensor using the SIP protocol.

## 1. The HomeSIP project – Home Automation with SIP

### 1.1 Introduction

For some years, house automation has become part of modern home landscape to enhance users comfort, security or both. Indeed, most remote-controlled devices such as temperature sensors or alarm systems have invaded houses, but these embedded systems need to be gathered: that is the point of house automation. Linux software is known to be portable and open source: these are the major reasons why house automation device developers recently try to implement embedded Linux systems. The HomeSIP Project consists in portability of embedded Linux software applied to temperature sensor based on SIP protocol. First, the project and its objectives will be introduced; then the previous work on it will be quickly presented. Finally, our approach will be detailed, namely the scratchbox installation, the development of SIP based communication program and the graphical user interface.

### 1.2 HomeSIP project overview

The HomeSIP project is collaboration between IXL, Labri and ENSEIRB. It consists in setting up a Hardware/Software platform based on the SIP protocol (Session Initiation Protocol). Namely, the main objective is to setup a sensor network for Home Automation that will be monitored remotely by smart devices using SIP as a container for collecting data.

The main three components of the HomeSIP project are:

- ✓ a hardware platform made of sensors and actuators connected to Linux embedded systems with network capabilities;
- ✓ specific software on Linux embedded systems implementing the SIP protocol to communicate with the sensors;
- ✓ a new DSL language (Domain Specific Language) made to develop new services on the HomeSIP platform.

The sensor hardware gateway which is operational today is based on commercial hardware:

- ✓ An ARM9 SBC board running embedded Linux with I/O connections (USB, serial line) and an Ethernet interface to act as a SIP gateway.
- ✓ An i-button DS1920 temperature sensor from Dallas Semiconductors connected via serial line connection.

Applications have already been developed to drive sensors on the ARM9 SBC board and to collect data from sensor via SIP messages on remote Linux systems.

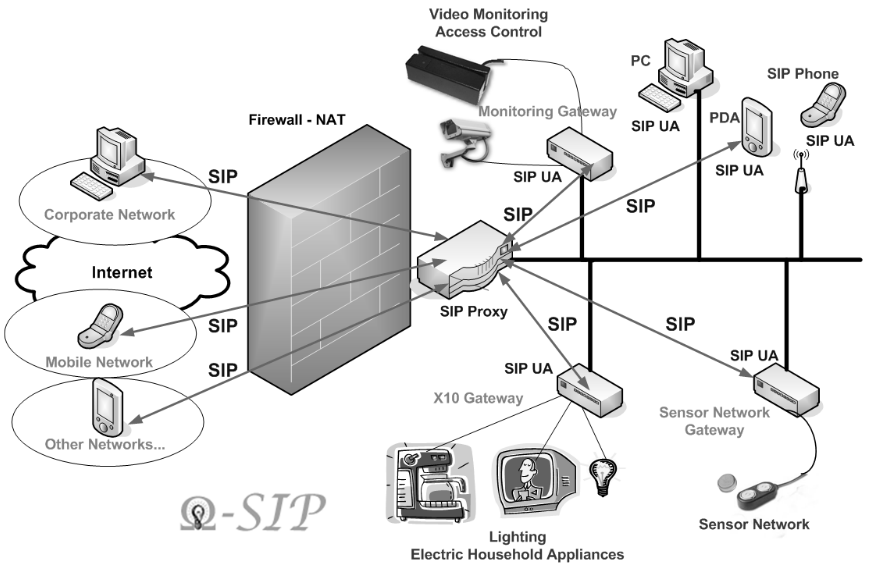


**Figure 1** –The HomeSIP hardware platform [1]

## 1.3 The choice of SIP protocol

The Session Initiation Protocol (SIP) is an application-layer control (signaling) protocol for creating, modifying, and terminating sessions with one or more participants.

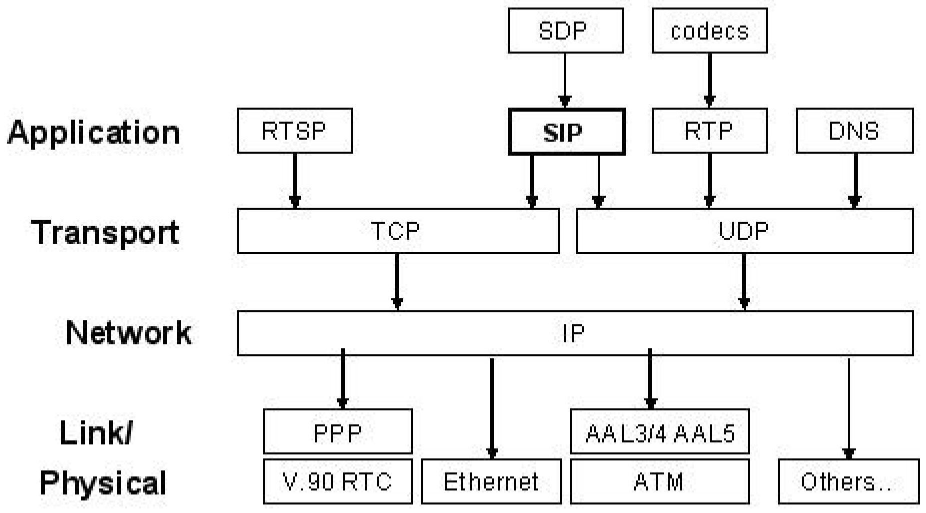


**Figure 2** – SIP protocol in the TCP/IP constellation [1]

SIP is a very flexible ASCII oriented protocol and can be used with any kind of data or transport network. Nowadays, it is mostly used for Voice over IP (VoIP) or instant messaging but it also has several qualities for home automation:

- ✓ it supports abstract addressing/naming;
- ✓ it provides security: both authentification/authorization and privacy;
- ✓ it supports different communication mechanisms for devices;
- ✓ it provides a flexible payload capability based on MIME types;
- ✓ it works with different in-home networking technologies : CPL, X.10, CAN bus;
- ✓ it can reuse existing SIP infrastructures for new services.

Those qualities, alongside with a very small memory footprint, makes SIP the best choice for home automation compared to other protocols such as SMNP or HTTP.

The SIP stack that was chosen to develop applications is the oSIP stack and its extension API eXosip made by Aymeric Moizard to simplify SIP application writing. It is written in C language, is very portable and has very low footprint. oSIP supports many transport protocols such as TCP, UDP, and TLS (Transport Layer Security: formerly SSL).

**1.4 Objective: a user-friendly graphical interface on PDA Nokia n770 to check the temperature**

The main goal of the HomeSIP project consisted in writing an application on a PDA (Personal Digital Assistant) with a graphical interface to get the temperature from the i-button sensor using the oSIP stack and eXosip libraries. The sensor remained connected to an ARM9 SBC board with embedded Linux and an Ethernet network interface. The PDA was a Nokia n770 based on a Linux system and was connected via USB to a Linux Personal Computer acting like a router between USB and Ethernet Network. The client software was developed on the Nokia n770 and the server on the ARM9 board. In order to develop the GUI (Graphical User Interface), the maemo SDK, a development platform made by Nokia for its internet tablets which uses GTK+ libraries and the Hildon application framework, was used.

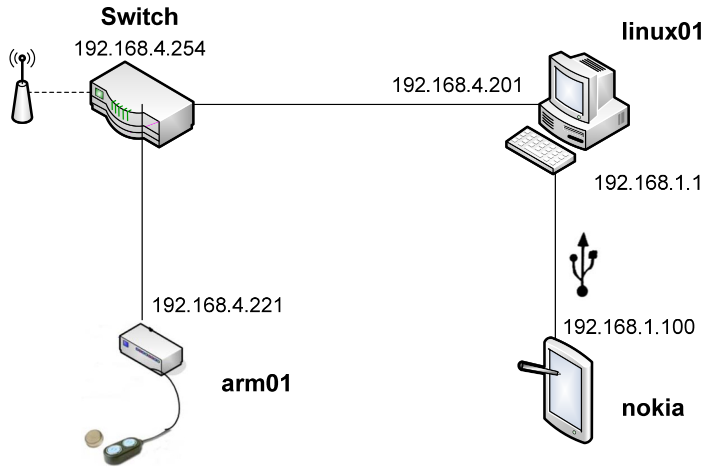


**Figure 3** – Hardware architecture of the platform

## 2. Previous work on HomeSIP project

Because the HomeSIP project is a heavy one, requiring knowledge on embedded devices, programming and network protocoles, a group of Electronics third-year students started working on it last year. The first part of the project consisted in getting the temperature of the sensor connected to the smart device ARM9 on the PC using a local network between the machines. Therefore, the protocol SIP was implemented to set-up the communication using the OSIP stack. It was chosen because it is the most appropriate stack regarding the limited resources of the smart device. The library was downloaded and compiled differently depending on CPU architecture. The PC had a X86 architecture whereas the smart device had an arm one. Differents compilers were used, gcc for the PC and arm-linux-gcc for the smart device.

Then, the communication programs were implemented:
- ✓ « send_im » sent the temperature to the PC instantly without acknowledgment. Applications like Joshua or Gaim were used on the PC to receive these messages.
- ✓ « send_notify » was a more reliable application. A session was started between the server (the smart device) and the client (the PC). Initial subscription requests and responses were exchanged between the machines in order to set-up the communication before temperature was sent.
- ✓ « send_subscribe » was the application installed on the PC that started the communication with the server in order to get the sensor temperature.

The first two applications were compiled with the arm compiler while the last one was compiled for the PC architecture. Here is the compilation command for the arm architecture:

```
    arm-linux-gcc   -O2   -g   -I/home/guest/arm/include   -L/home/guest/arm/lib   -
DENABLE_TRACE $1.c  -o $1 -leXosip2 -losip2 -losipparser2 -lpthread
```

## 3. The work done

### 3.1 The development environment

#### Cross-compilation

Developing applications using the Nokia n770 device was not possible for two reasons. The first one was that the device had limited resources (memory, disk space,…). The other one was that the n770 was an Internet Tablet. Thus, it did not have the necessary tools for developping and compiling. An open source development platform provided by Maemo was used instead. Indeed, it is built from components widely used in open source desktop and mobile systems. The Maemo software development kit (SDK) contains the tools needed to create and port integrated applications, replicating the Internet Tablet environment in the PC which has a different CPU architecture from the n770 one. Therefore, a cross-compilation toolkit was required to compile executable code for the ARM9 platform using the PC. This was accomplished using a cross-compiler toolchain and cross-compiled libraries and specifying these in the Makefile that built the software.

#### Scratchbox

Scratchbox, a cross-compilation toolkit, was used because it is designed to make embedded Linux application development easier by integrating rootstraps for different CPU architectures.

There are different versions of Maemo SDK. The one used in this project was the Maemo 2.2 'gregale' which targeted the n770. This version required the installation of Scratchbox Apophis R4 or later.

The first step was to download the scratchbox packages and install them. After that, the environment could be started but there was no compilation target. Therefore the next step was to download and install the Maemo SDK rootstrap. A cross-compilation target for ARM, the n770 CPU architecture, was created on the scratchbox and configurated by selecting the adequate compiler and CPU transparency, as shown on Figure 4. The downloaded rootsrap packages were extracted to set-up the environment.

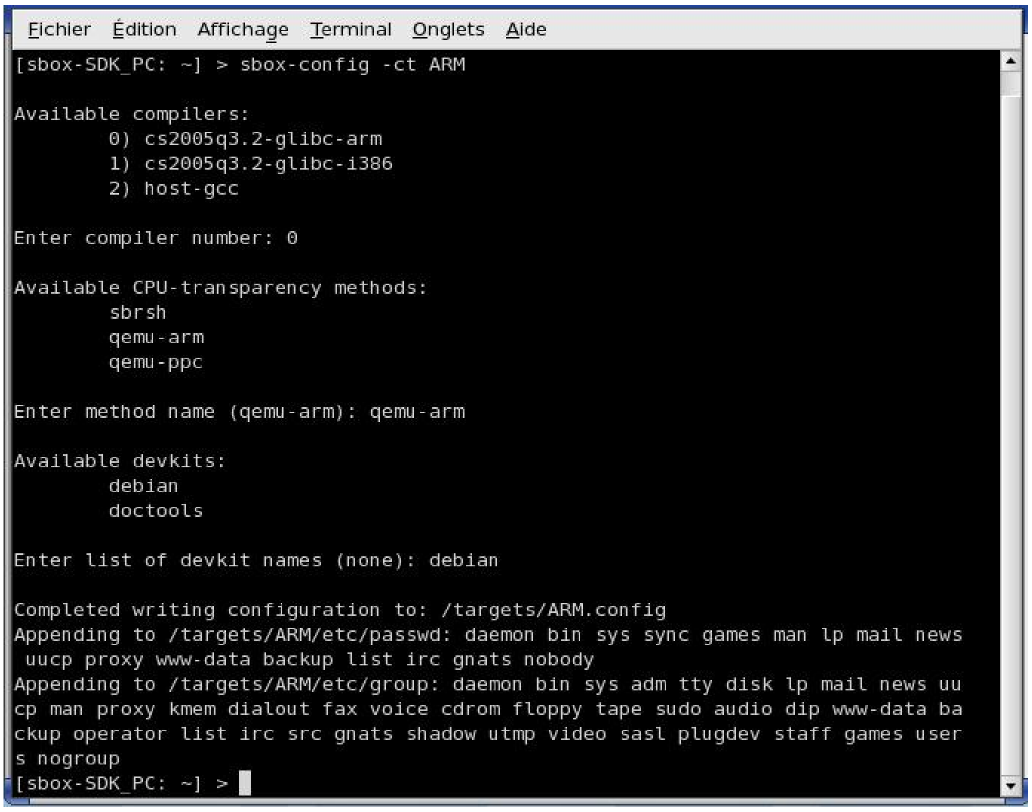
#### Xephyr

Although the Maemo development environment can generally start compiling Maemo applications for n770 device after installing the ARM rootstrap, a graphical X server was required to be able to see and test the applications. Xephyr was recommended because it was pre-installed in the rootstrap. It is a drive-based X server which targets a window on a host X server as its frame buffer.

In order to work properly on the n770 when loaded, the library had to be cross-compiled on the scratchbox platform. The compilation generated binary headers, where functions, data structure, and library files were defined. Indeed, when SIP applications are compiled or executed, the SIP funtions and data structures are loaded from the header and library files. For compilation, the paths were declared manually with the command:

```
    gcc -IoSIP/include -LoSIP/lib -leXosip2 -losip2 -o $1 $1.c `pkg-config --
cflags gtk+-2.0 hildon-libs` -ansi -Wall `pkg-config --libs gtk+-2.0 hildon-libs`
```

For the execution, the files had to be copied into the appropriate directories of the Nokia Internet Tablet without losing their dependencies. Therefore, they were archived together using tar program. Then they were transmitted to the device via a local network between the developing machine and the Nokia device. It allowed to send applications using the scp command (Securised copy) or to configure the device parameters using the ssh (Secure Shell) connection. When copied into the Nokia device, the files were extracted and moved to the corresponding directories: the /lib directory for library files and /include directory for header ones.



```
[sbox-SDK_PC: ~] > sbox-config -ct ARM

Available compilers:
        0) cs2005q3.2-glibc-arm
        1) cs2005q3.2-glibc-i386
        2) host-gcc

Enter compiler number: 0

Available CPU-transparency methods:
        sbrsh
        qemu-arm
        qemu-ppc

Enter method name (qemu-arm): qemu-arm

Available devkits:
        debian
        doctools

Enter list of devkit names (none): debian

Completed writing configuration to: /targets/ARM.config
Appending to /targets/ARM/etc/passwd: daemon bin sys sync games man lp mail news
 uucp proxy www-data backup list irc gnats nobody
Appending to /targets/ARM/etc/group: daemon bin sys adm tty disk lp mail news uu
cp man proxy kmem dialout fax voice cdrom floppy tape sudo audio dip www-data ba
ckup operator list irc src gnats shadow utmp video sasl plugdev staff games user
s nogroup
[sbox-SDK_PC: ~] > 
```

**Figure 4** – Rootstrap packages for setting up environments in Scratchbox

### 3.2 Communication by SIP between the client and the server

The goal of the project was to make the client (nokia) and the servers (arm01) communicate. In this part, algorithms and communication by SIP will be detailed, without considering the graphical aspect.

Firstly, the client had to contact the server. In its subscription message, the client indicated how long the exchanges had to last. Once the communication was established, the server regularly sent the temperature it collected from the iButton until the end of the timer. The exchange of SIP messages are given in figure 5.

So that communication could be established, two simplified programs were developed, for each side.
In the following lines, some parts of the programs are detailed. In order to simplify the reading, some comments and the error tests were removed.

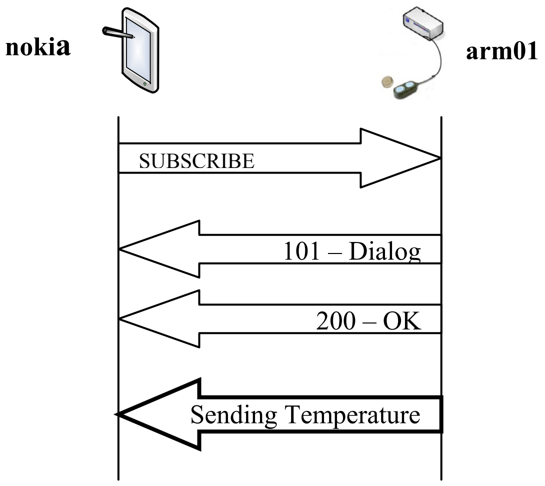


**Figure 5 –** Exchange of SIP messages so that the client gets the temperature from the server

**Nokia**

The program required three arguments:

- ✓ Host SIP address: `sip:root@target`
- ✓ Target SIP address : `sip:root@host`
- ✓ Communication duration: `Expiration_time.`

The SIP protocol was initialized with the client IP address: 192.168.1.100.

```
eXosip_init ();
eXosip_listen_addr (IPPROTO_UDP, NULL, 5060, AF_INET, 0);
eXosip_force_masquerade_contact ("192.168.1.100");
```

Then the subscribe request was built and sent:

```
char buf[256] = "Subscribing to the DS1920 sensor";
eXosip_subscribe_build_initial_request(&message,argv[2],argv[1],NULL,
"presence", 1200);
*(message->cseq->number) = atoi(argv[3]);
osip_message_set_body (message, buf, strlen (buf));
osip_message_set_content_type (message, "text/plain");
eXosip_lock ();
eXosip_subscribe_send_initial_request (message);
eXosip_unlock ();
```

While the expiration time was not reached, the client waited for a SIP message and tested its type. If it was the sending of the temperature, the latter was printed.

```
    event = eXosip_event_wait (2,500)));
    eXosip_automatic_action ();
    switch (event->type) { […]
       case 27:
       osip_body_t *body3;
       osip_message_get_body (event->request, 0, &body3);
        if (body3 != NULL && body3->body != NULL)
              printf ("%s\n", body3->body);
    […]}
```

**Arm01**

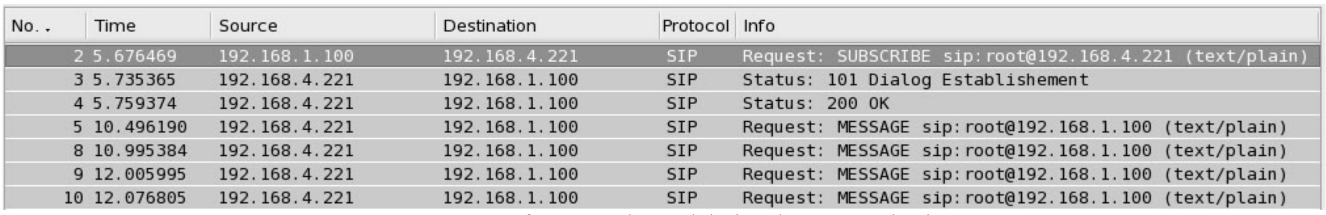
After initializing the SIP protocol, the server (IP address: 192.168.4.221) had to wait for a SIP subscription message from the client to establish a connexion. It got the client SIP address and the expiration time and then, it started sending messages with temperature value in the message body. This temperature was obtained thanks to another external function (gettemp_ds1920) which had already been developed. More precisely, a function put the temperature in a buffer and then the main function included it in the request body.

```
osip_from_to_str(event->request->from,&to);
osip_from_to_str(event->request->to,&from);
from=strsep(&from,”>”)+1;
to=strsep(&to,”>”)+1;
eXosip_message_build_request (&message, "MESSAGE", to, from, NULL);
temp();
osip_message_set_body (message, buffer, strlen (buffer));
osip_message_set_content_type (message, "text/plain");
eXosip_lock ();
eXosip_message_send_request (message);
eXosip_unlock ();
```

After cross-compiling the two programs (under scratchbox for the client), their execution gave what is written in Table 1. Thanks to Wireshark, as shown on figure 6, the SIP messages, exchanged during the communication, matched what was expected.

**Table 1** – *Programs executions on the client and on the server*

| Nokia (client) | Arm01 (server) |
|---|---|
| `~#./test_im  sip:root@192.168.1.100 sip:root@192.168.4.221 30` | `# ./send_notify_im` |
| `Annonce 1xx` | `200 OK Response Building OK` |
| `Annonce 200 OK` | `200 OK Response Sending OK` |
| `01000800BC63CA10      28.0` | `Temperature : 28.0` |
| `01000800BC63CA10      28.1` | `Temperature : 28.1` |
| `01000800BC63CA10      28.1` | `Temperature : 28.1` |
| `01000800BC63CA10      28.2` | `Temperature : 28.2` |
| `01000800BC63CA10      28.3` | `Temperature : 28.3` |



| No. | Time | Source | Destination | Protocol | Info |
|---|---|---|---|---|---|
| 2 | 5.676469 | 192.168.1.100 | 192.168.4.221 | SIP | Request: SUBSCRIBE sip:root@192.168.4.221 (text/plain) |
| 3 | 5.735365 | 192.168.4.221 | 192.168.1.100 | SIP | Status: 101 Dialog Establishement |
| 4 | 5.759374 | 192.168.4.221 | 192.168.1.100 | SIP | Status: 200 OK |
| 5 | 10.496190 | 192.168.4.221 | 192.168.1.100 | SIP | Request: MESSAGE sip:root@192.168.1.100 (text/plain) |
| 8 | 10.995384 | 192.168.4.221 | 192.168.1.100 | SIP | Request: MESSAGE sip:root@192.168.1.100 (text/plain) |
| 9 | 12.005995 | 192.168.4.221 | 192.168.1.100 | SIP | Request: MESSAGE sip:root@192.168.1.100 (text/plain) |
| 10 | 12.076805 | 192.168.4.221 | 192.168.1.100 | SIP | Request: MESSAGE sip:root@192.168.1.100 (text/plain) |

**Figure 6** – SIP frames exchanged during the communication.

With these two programs, temperature from the Nokia could be displayed in a terminal. The next step was to integrate the client program in a user interface.

## 3.3 Programming the Graphical User Interface (GUI)

### Maemo Software Development Kit

Maemo is an open source development platform for Linux-based handhelds, built from widely used open source components with additional features.

With the maemo sofware development kit (SDK), it is possible to run ready-made applications on the maemo platform using a Linux PC, and create maemo UI applications which can be tested and debugged inside the maemo SDK with a normal Linux PC ([2]). Thus, software development for embedded devices is as easy as for desktop PCs. That is one of the main reasons why maemo environment was chosen to develop an interface for the Nokia Personal Digital Assistant (PDA).

Running applications inside the development environment is easy and, most importantly, they are run in the real Linux environment instead of an emulator.

### Hildon libraries

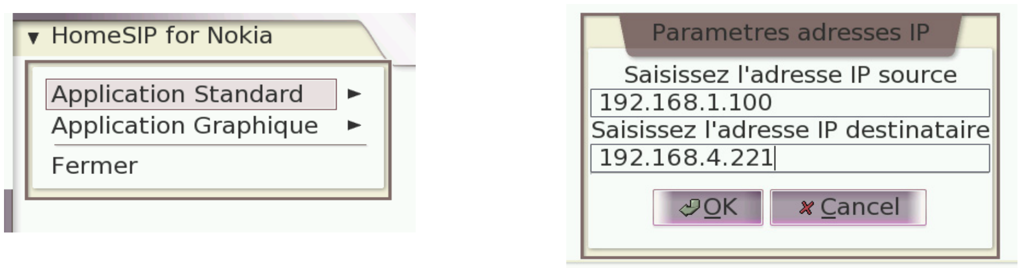
Hildon is a graphical user interface (GUI) designed for small mobile devices and especially for Nokia n770 interface. Most GTK+ widgets and methods work unaltered in the Hildon environment, but to make the applications fit in the maemo UI environment, some source code changes for plain GTK+ applications are required.

### Oriented Object Programmation with GTK+ and C

Despite being suggested by Maemo 2.1 Tutorial, Hildon libraries revealed to be unsufficient to write a complete GUI achieving what was expected, namely collecting and displaying temperature caught by the i-button sensor. Indeed, it was necessary to get first used to oriented object programming using GTK+ libraries, which is the most basic developed tool kit for GUI. The second main problem was C programming: obviously it enhances portability, but using C language is far heavier than using any other oriented object language such as C++, Java, etc. Indeed, each time a widget or a frame needs to be added to the interface, an amount of code lines must be added to the program, which is far from being convenient.

### A simple and practical user interface

A simple and practical GUI was implemented: a single-level menu, information boxes to catch data like IP addresses or time-out duration, and a structured display application. A few examples are given below:

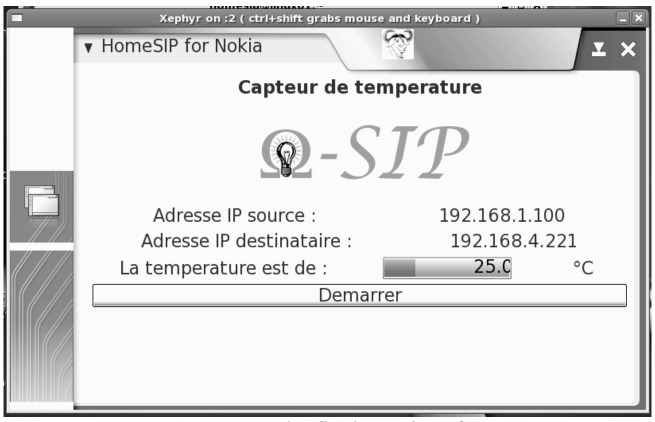


Saisissez l'adresse IP source
192.168.1.100
Saisissez l'adresse IP destinataire
192.168.4.221

**Figure 7** – A single-level menu and an information box

### Adapting the instant messages program for Armel

The final and the most important part of the project was the adaptation of the instant message program written by the students who had previously worked on HomeSIP project. Indeed, there were a few problems concerning compiling programs using eXosip libraries: it only worked when directly compiling for ARMEL, not when testing on the PC.



Adresse IP source : 192.168.1.100
Adresse IP destinataire : 192.168.4.221
La temperature est de : 25.0 °C

**Figure 8** – The final user interface

## 4. Conclusion

Even if we have had some difficulties to use oSIP and eXosip libraries on the Nokia n770, we have finally managed to make a functional user-friendly graphical user interface that can be used by anyone. Since this interface is written with Hildon (based on GTK+) it could easily be ported to other Linux systems. Moreover, this project has shown that SIP is a reliable protocol for Home automation. A good amelioration to the system would be to use a Wifi interface instead of the USB interface to communicate with the PDA.

## References

[1] P. Kadionik, « Le projet HomeSIP : la domotique avec le protocole SIP », Linux Magazine, HS 25, april-may 2006
[2] Maemo 2.2 Tutorial, http://maemo.org/development/documentation/tutorials/Maemo_2.2_Tutorial.html