GeCSen – A Generic and Cross-Platform Sensor Framework for LocON

Mitch De Coster and Steven Mattheussen
Dept. of Applied Engineering
Artesis University College
Antwerp, Belgium
mitch.decoster@student.artesis.be
steven.mattheussen@student.artesis.be

Martin Klepal
Centre for Adaptive Wireless Systems
Cork Institute of Technology
Cork, Ireland
martin.klepal@cit.ie

Maarten Weyn and Glenn Ergeerts
Dept. of Applied Engineering
Artesis University College
Antwerp, Belgium
maarten.weyn@artesis.be
glenn.ergeerts@artesis.be

***Abstract*—In this paper, we present a generic and cross-platform sensor framework for the LocON location and sensor middleware. This framework is developed using C++/Qt. Sensor information is rapidly gaining importance in automating processes and ubiquitous computing, and so it is for projects like FP7 LocON where the main goal is to integrate embedded location systems and embedded wireless communication in order to manage and secure large scale environments. Accessing sensor information mostly requires platform specific code, but for merging this information and sending it over the internet or displaying it on a device, Qt provides cross-platform libraries. Our sensor framework, called GeCSen, also comes in the form of a library that can load platform specific sensor plugins and is able to communicate with the LocON middleware. The framework acts as a cross-platform layer which sends the sensor information to the LocON middleware. This framework enables all kinds of sensors on a wide range of devices to be used by the LocON platform and thereby adds substantial value to the FP7 LocON project.**

***Keywords*-monitoring, localisation, embedded devices, smartphone, sensors, locon, cross-platform**

I. Introduction

Nowadays, sensor data is widely used for many applications, for example, monitoring patients, personnel and equipment in hospitals, monitoring athletes to optimise training methods, etc. To do monitoring we need a selection of sensors such as location sensors, temperature sensors, accelerometers, heartbeat sensors, etc. and some basic logic to parse the sensor data. This data is used to follow someone or something and draw conclusions on which certain actions can be based, for instance, calling an ambulance and automatically give GPS coordinates when someone is having a heart failure. There are different kinds of monitoring but in general we can conclude that there is environmental monitoring to control an area, object monitoring to track a person or object and process monitoring to monitor the proceedings of a process. All situations require different approaches. We are mostly focused on monitoring moving objects.

Currently, monitoring is mainly done by what are called wireless sensor networks [1]; these networks typically consist of a variety of sensor nodes with a small battery, microcontroller and radio transmitter that collects, processes and communicates sensor data. All sensor nodes work together to monitor an environment whereby all collected sensor data reaches a central sink. The sink acts as a link between the sensors and the application. With GeCSen we aim to use computers, embedded systems and smartphones. Many of these devices already have the means of communication and are equipped with various onboard sensors, for instance, most smartphones have a subset of the following: GSM/UMTS, Wi-Fi, GPS, accelerometers, Bluetooth, etc. and can also be extended with various other sensors to suit your needs. Using these kinds of devices comes at the cost of considerably higher power requirements, but also gives enormous processing power in the node itself.

Most applications of sensor usage today are custom developed with industrial usage as target, thus they are specifically built for their needs and only work with specific hardware. Some commercial applications, for example for athletes, are also available, but they are mostly device specific and all use different approaches of which practically none are generic. Notwithstanding, there are already some efforts in making generic sensor frameworks, for example the S60 Sensor Framework of Nokia, the Moblin sensor framework and mSense [2]. Unfortunately they are mostly platform specific, and no more than merely an API for sensors, thus there is an urgent need for a generic way to access sensors used for monitoring.

This paper describes a generic and cross-platform sensor framework, developed in C++/Qt which is easily extendible with plugins to support all kinds of sensors on the majority of platforms and architectures. It thus provides a consistent method of accessing sensor hardware by adding a cross-platform abstraction layer above the APIs described in the previous paragraph, so the plugins are a platform specific wrapper around the APIs which can be loaded in the sensor framework. We also send the sensor data to the middleware of the FP7 LocON project [3], thus we are able to use the LocON data collection and data mining

 ISBN: 978-1-61208-100-7

abilities, together with its localisation fusion engine. Using the LocON platform gives us the opportunity to thrive with the success of the FP7 LocON project, but GeCSen itself also provides substantial value to LocON as it makes it very easy to enable all kinds of sensors on various devices for the LocON platform. We provide the framework itself as a library which can be used statically or dynamically in a host application. The host is able to control GeCSen and use the collected sensor information to, for example display it in a GUI.

The remainder of the paper is organized as follows. In Section 2, we discuss cross-platform aspect of GeCSen and why it is important. The architecture of the framework is described in Section 3. Section 4 goes deeper into the features of GeCSen. In Section 5 we discuss the test cases used to demonstrate the usefulness of the framework, followed by a comparison in Section 6 which shows the advantage of using GeCSen. Future work is discussed in Section 7 and finally, section 8 concludes the paper.

II. Cross-Platform

GeCSen has a broad target group, thus it needs to be able to run on as much platforms as possible. It is not a trivial task to develop a truly cross-platform application, because every platform uses its own libraries and has its own platform definitions. Writing cross-platform applications, basically means having to choose between two major programming languages, C++/Qt and Java. Java is a platform independent language which uses a virtual machine to run Java programs. Qt, on the other hand, is a toolkit written in C++ that uses the same APIs for different platforms. Choosing one over the other essentially means choosing a subset of platforms that you want to support. The main advantage of C++ over Java is a more efficient use of processor cycles and memory and it allows us to program closer to the hardware[4],[5]. The main disadvantage of C++ is that you have to cross-compile for each platform while with Java you do not.

Applications that use GeCSen are mainly focused on mobile and embedded platforms. The market for embedded devices is relatively stable; the two mostly used operating systems are Windows CE and various embedded Linux distributions. The mobile market, on the other hand, is constantly fluctuating; software platforms come and go. As Canalys [6] concludes the most used operating systems currently are Symbian and Blackberry but they also conclude that both Apple iPhone and Google Android are growing very fast, mostly at the cost of Symbian and Windows Mobile. On top of that, every mobile OS has its own set of APIs and most of them are very restricted. This makes developing an application that works on every one of them nearly impossible.

There is also some research going on about cross-platform mobile development which results in a number of very different approaches. Cha, Bernd and Du [7] propose a comprehensive mobile application framework to support interoperability and mobility of mobile application development and operation. Choi, Yang and Jeong [8] suggest an application framework for writing cross-platform mobile applications in Java. PhoneGap [9] is a project that provides a framework in which you can develop programs using simple HTML, CSS and JavaScript. It supports iPhone, Android and Blackberry. MoSync [10] , another project, provides a codebase with which you can program cross-platform in C/C++ for Java ME, Symbian S60, Windows Mobile and Moblin and they are working on Android, iPhone and Maemo. Unfortunately these methods are still too restricted for us to use and don't allow us to program for desktops and laptops, as well as mobile devices.

We have chosen C++/Qt, because although Java is supported by more smartphones than C++/Qt, most smartphone manufacturers have their own restricted APIs. By using C++/Qt a significant part of the smartphone market will still be supported. In addition, Qt is fully open-source; thereby it has a large developer community that thrives to port Qt to every possible platform. It is also gaining more and more attention since Intel and Nokia announced that they are strongly working together to make Qt the default in cross-platform development. GeCSen is tested and works on various x86/x64 and ARM architectures in combination with the following platforms: Windows, Windows Mobile, Windows CE, Android and various Linux distributions, including Ubuntu and a number of OpenEmbedded variants.

III. Architecture

The GeCSen architecture, which is shown in Figure 1, shows a generic way to enable sensors on a device to communicate with the LocON platform. Previously for every different type of device which had to be connected with LocON, or for adding an extra sensor to the LocON client application, it had to be partially, if not completely rewritten. Using this sensor framework the same code can be reused, one only has to compile it for the specific device and develop plugins to use the sensors. As a result, it becomes very easy to add extra sensor devices to the LocON platform. The plugins form a hardware and platform specific layer between the cross-platform framework and the sensors, thus they are responsible for managing the sensors and sending the sensor data to the sensor framework in a uniform way. LocON is working on the standardisation of the data protocol; therefore, the framework has to translate the sensor information, obtained by the plugins, into LocON compatible messages. This is done by using the LocON protocol library [11]. These messages, which can be extended data messages or position messages, are then sent over the network to the LocON platform. This sensor information is very useful to help achieving the main goal of the LocON project, namely securing large scale environments, but also gives endless other possibilities to LocON.

 ISBN: 978-1-61208-100-7

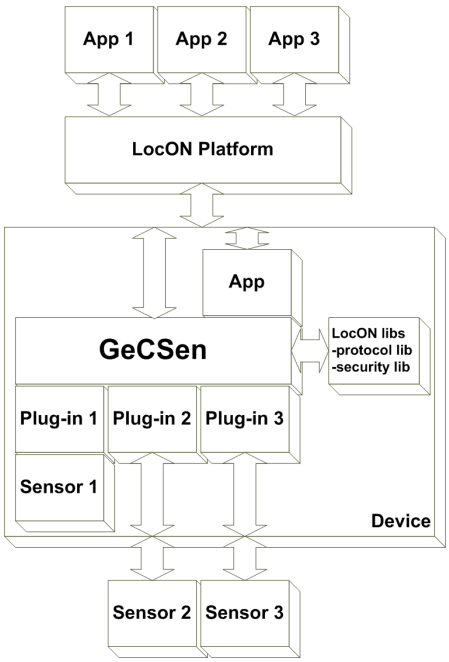

Figure 1. Architecture of the sensor framework

The architecture is in essence a client-server architecture. GeCSen is a client which runs on a device such as a computer, an embedded device or a smartphone. The goal of the GeCSen client is to manage sensor plugins, to collect sensor data from them, to process that data and to send it to the LocON platform. The LocON platform is the server; it is responsible for collecting and processing the sensor data and for seamlessly combining position data. The application on top of the LocON platform is essentially another client that queries the platform for sensor data and position updates so it can be represented in a graphical and useful way or it can be used for further processing.

Figure 2 shows how GeCSen, its plugins and its host applications fit in the OS architecture. It runs on top of the Qt framework and is able to access the system hardware trough the APIs, services and libraries.

IV. Features

A. Plugin System

The plugin system is based upon the Qt Plugins API. The plugins are in essence dynamic libraries that are loaded by the GeCSen framework using a declared interface. This interface is a class containing solely virtual functions that are then used to communicate with the plugins. Every plugin has to have an initialisation and configuration function. The initialisation function, which is the first call to the plugin, is used to initialize the necessary steps, for example, initialise the sensor and start a timer for reading its information. The configuration function is used to configure the plugin and

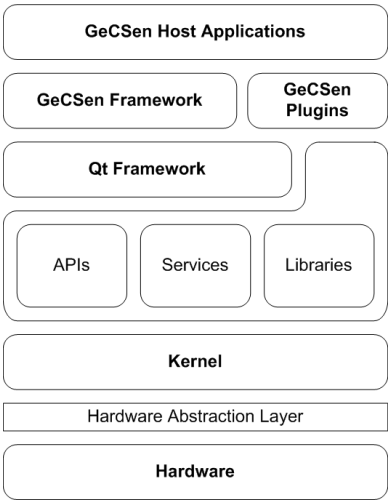

Figure 2. GeCSen on top of OS Architecture

is invoked by GeCSen whenever it receives configuration messages from LocON that are destined for the plugin.

1) Pushing mechanism: Generally plugins are called by the application, accordingly, in our case the framework has to poll the sensors for data. Following the fact that the plugin has to send the sensor information when it is appropriate, for example when a button is pressed, a pushing mechanism had to be developed. When the framework has loaded the plugin and calls the initialisation function, two pointers to functions in GeCSen are given as arguments to the plugin. These two functions can be used to send position messages or extended data messages to the GeCSen framework.

2) Configuration messages: One of the features of LocON is sending configuration messages which can be used for various purposes, GeCSen also supports these configuration messages. By defining and linking them to a sensor in the GeCSen configuration XML, this defined message will be sent to the sensor plugin by using the virtual configuration function in the plugin interface. These configuration messages can be used to set the update rate of a sensor or to manage an actuator for example.

B. Sensor Data Collection

After the plugins are loaded and initialised, they start sending sensor data to the framework by using the function pointers we discussed in the previous paragraph. For extended data messages there are a number of data types available that can be sent which are defined by the LocON protocol and specified for each sensor in the configuration XML. All sensor data is sent as a list of generic pointers, so there is no need for conversion to a certain data type. By using a list we enable the plugin to send multiple values within the same message, for instance multiple MAC addresses of visible Bluetooth devices. Together with these data, the name of the sensor and the plugin is also sent.

 ISBN: 978-1-61208-100-7

Subsequently the data is processed by the sensor handler which checks if the sensor that is sending is allowed to send data and then converts the data into the right format for further processing by the LocON communicator or for usage in the GeCSen host application.

C. LocON Communicator

The LocON communicator uses the LocON protocol and security libraries together with the Qt network libraries to make a connection with the LocON platform and send the right packets accordingly. GeCSen needs to authenticate itself to the LocON platform before it can send any packages. Therefore the communicator sends a handshaking message to the LocON platform. This handshaking message contains a LocON packet with only the subsystem id in it. With this message GeCSen asks to initiate the authentication process. In return LocON sends an open authentication message containing an encrypted blob and a signature blob. Thereafter the paths to the public LocON key and private device key together with the subsystem id are stored in an internal security structure. This structure is used to process the open authentication message received from LocON. When this message is valid an acknowledge message is sent back to LocON to complete the authentication process. After this process GeCSen is able to send signed, and in the future encrypted, messages to LocON. When the connection is lost the communicator will automatically retry to establish a new connection every ten seconds.

Using the data received from the sensor handler a LocON packet is being build. This packet will be filled until the update time interval has passed or when the maximum package size is reached. Thereafter, it is sent to the LocON middleware and the framework starts filling up a new packet.

D. Host Application

Our sensor framework is provided as a library, meaning that it needs a host application to be loaded and initialised. For this purpose GeCSen provides some functions. The most important function is the constructor with parameters to tell GeCSen the location of the configuration file, the plugins directory, the ssl keys for authenticating with LocON and where the log file should be placed. The constructor initialises and starts GeCSen. Some other functions are provided to start and stop GeCSen and to start and stop logging. Qt host applications can also use the signals, of the Qt signal/slot mechanism, we provided for when new extended data, position data or configuration messages arrive. This allows host applications to use sensor data, for example in a GUI on the device itself. Host applications can also be implemented as a console application or even as a service.

E. Configuration

GeCSen can be configured with a XML file similar to the SubsystemDefinitions.xml file used in LocON [**?**]. The GeCSen XML configuration first defines some configuration parameters like server IP and port, whether or not it needs authentication and the update interval in milliseconds. Then a single subsystem is defined which describes a subsystem in which configuration messages and extended data items are defined by a name and data type, exactly the same as in one of the subsystems in the SubsystemDefinitions.xml file of the LocON configuration. The last part of the GeCSen configuration file defines the plugins with their name and sensors that are coupled with extended data items and configuration messages. This allows us to enable or disable plugins and sensors using the configuration file. The configuration file is of great importance to GeCSen and is an agreement between LocON and GeCSen.

V. Test Cases

Various test cases have been developed to test the framework and demonstrate its usefulness. Similar applications already exist in various non-generic, non cross-platform implementations, thereby these test cases prove that we can achieve the same by using our sensor framework. Some test cases we describe here were developed as bachelor theses within our master's thesis, we supported their development and provided the necessary tools and documentation.

A. Remote Athlete Monitoring Application

The first test case is an application for monitoring sporting athletes [13]. GPS data from a HTC Touch Cruise smartphone is combined with the Zephyr HxM [14] external heart rate sensor connected via Bluetooth. Therefore two plugins are written, one to access the GPS sensor and collect the GPS coordinates and another to access the Bluetooth sensor and collect the heart rate values. This data is being presented in GUI applications on the smartphone itself and on top of the LocON platform. This test case should also work on other mobile phones that support Qt and have a GPS and Bluetooth chip, as long as the right plugins are provided.

B. Opportunistic Seamless Localisation Client

Another test case was to build an OSL client using GeCSen [16]. OSL uses the LocON protocol, but without authentication. We have developed a number of OSL compatible plugins such as a Wi-Fi plugin that measures the signal strength (RSSI) of all visible access points, a Bluetooth plugin that detects all Bluetooth devices in the vicinity and a standard GPS plugin. Other possible plugins that we did not provide are a plugin for GSM, an activity plugin that measures the time since your last keystroke and one for step detection with accelerometers. GeCSen, together with a set of OSL plugins is able to act as an OSL client. All these plugins are developed for Windows, Windows Mobile, Linux and Embedded Linux.

 ISBN: 978-1-61208-100-7

C. Internal Social Networking Application

This test case is a social networking application for Windows [17], developed in C#. With this application, friends can locate each other using Wi-Fi based localisation. This application is meant for internal use at our campus to allow students to check if teachers are in their office. To make this possible the application uses GeCSen to send the RSSI values and MAC addresses received from a Wi-Fi plugin to the OSL server using the OSL client implementation described earlier. The OSL server in turn calculates the position of the device sending the Wi-Fi information. For this test case a C# wrapper for GeCSen was developed.

D. Demo Application

During our master's thesis some other simple plugins were developed in order to test GeCSen. This includes a dummy plugin which has two sensors that send dummy values at different intervals, this dummy plugin works on all supported platforms; a CPU plugin that measures CPU usage for Windows, Linux and Embedded Linux and a battery plugin that measures remaining battery life in percent for Windows, Linux and Embedded Linux (ACPI and APM). These plugins are then used together with the other plugins that were previously discussed. For this test case a simple test GUI for the device and an interactive application on top of LocON where you can collect and visualise all the information from the connected devices were developed to demonstrate the advantages of GeCSen in an interactive demo.

VI. Comparison: GeCSen versus OSL client

There are already some systems developed that communicate with and use the LocON platform. For example, the Opportunistic Seamless Localisation System (OSL) client, this is the client that communicates with the OSL server, which uses the LocON protocol. This server combines all sensor data to do opportunistic localisation and in turn sends the location data to the LocON platform [16]. There are a number of OSL clients developed for various platforms including Windows, Windows Mobile and embedded Linux. These clients consist of different implementations to talk to the hardware and to communicate with the OSL server. The OSL client for Windows is developed using C# whereas the version for embedded Linux is using Qt Extended and there is both a C++/Qt as a C# version for Windows Mobile. As you see these are four totally different implementations for achieving the same purpose.

However, the underlying principle is the same for all implementations. When the sensors are being read their data, such as access point information and accelerometer data, is sent to the client controller in a non-generic manner. When new sensors need to be added to the OSL client the client needs to be partly rewritten and compiled. GeCSen is able to do the same job as the OSL client but in a generic manner, adding a new sensor is as easy as developing a small new plugin and adding it to the configuration. The plugin is then loaded by the framework the next time GeCSen starts. This is done without doing any changes to the framework. We believe that GeCSen adds tremendous value for both developers and users of such applications like opportunistic localisation.

VII. Future Work

GeCSen still has to be tested thoroughly with a stress test over a long period of time on various devices as some optimisation of our code might be needed. For example, the device id is currently generated by scanning for network interfaces, chosing the best MAC address and storing it in a file for later usage. In the future when every device has its own certificate, it seems better to use a hash of that certificate as device id, because this is much more difficult to spoof compared to a MAC address. Improvements can also be made to the parser of the configuration XML file. We previously used XML schema validation but it added too much overhead to GeCSen because the Qt library for XML schema validation is too large. This means that our current parser is not completely fail-safe.

The sensor framework should also be tested on more platforms and devices that Qt is ought to work on. For example, the Symbian S60 platform on which we started, but did not succeed within the time that was anticipated. This was mainly because of the fact that we did not have a S60 smartphone to develop on and the standard emulator from the Symbian S60 SDK did not fulfill our needs. Some other mobile platforms on which GeCSen should also work, but are not yet tested include Maemo 5, MeeGo and Symbian 3. The Qt community is currently also working on the Qt Mobility project [18], this is a set of APIs that make it easier for developers to take advantage of mobile features such as using sensors. When this project is completed it could add tremendous value to the future of our project.

Another test case currently is in development as a bachelor thesis, called Smart Environment for Indoor Localisation and Evaluation [15]. The scope of this thesis is to solve the problem where patients who are in need for help still have to wait too long for treatment. Using this system doctors will be automatically warned when patients health conditions are abnormal. GeCSen will be used on a gateway to send sensor and localisation data to the LocON or OSL server for further processing. The GeCSen framework is expected to be included in the following up master thesis next year.

In order to keep GeCSen practically useful, a provisioning system would also be a valuable addition. We intended starting with the development of such a system, but due to time limitations we suspended the project. At last GeCSen should be promoted because of its simplicity and user friendliness. That is why we want to keep it alive by starting up a LocON alliance which recommends GeCSen to be used

 ISBN: 978-1-61208-100-7

as a standard way to connect devices with their sensors to the LocON platform.

VIII. Conclusion

Sensor readings add substantial value to projects like FP7 LocON. Nowadays, there is no generic and cross-platform tool for using sensors. GeCSen is a solution to this ever growing problem that occurs today. Combining various sensors as GSM/UMTS, Wi-Fi, GPS, Bluetooth, RFID and UWB can result in more accurate localisation systems. Other sensors as presence detection sensors can result in more secure environments. Whereas yet other sensors as heart rate sensors and temperature sensors can provide useful applications, for example to monitor sporting athletes or patients. We tried to port GeCSen to as many platforms and architectures we could get our hands on, and we reached even beyond our initial goal. Unlike existing sensor frameworks which are platform specific, GeCSen allows to expose generic sensor information from a broad range of platforms and devices to middleware like LocON, in an easy and uniform way.

References

[1] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "A survey on sensor networks," *IEEE Communications magazine*, vol. 40, no. 8, pp. 102–114, 2002.

[2] J. Krösche, A. Jakl, D. Gusenbauer, D. Rothbauer, and B. Ehringer, "Managing Context on a Sensor Enabled Mobile Device-The mSense Approach," in *2009 IEEE International Conference on Wireless and Mobile Computing, Networking and Communications*. IEEE, 2009, pp. 135–140.

[3] S. Couronné, N. Hadaschik, M. Faßbinder, T. von der Grün, M. Weyn, and T. Denis, "LocON–a Platform for an Inter-Working of Embedded Localisation and Communication Systems," *Proceeding of 6th Annnual IEEE SECON*, 2009.

[4] M. Kalle Dalheimer, "A comparison of qt and java for large-scale, industrial-strength gui development," Klarlvdalens Datakonsult AB, Tech. Rep., 2005.

[5] L. Prechelt, "An empirical comparison of c, c++, java, perl, python, rexx, and tcl," University of Karlsruhe, Tech. Rep., 2000.

[6] Canalys. (2010, May) Smartphone market analysis. [Online]. Available: http://www.canalys.com/pr/2009/r2009112.html

[7] S. Cha, J. Bernd, and W. Du, "Toward a Unified Framework for Mobile Applications," in *Proceedings of the 2009 Seventh Annual Communication Networks and Services Research Conference-Volume 00*. IEEE Computer Society Washington, DC, USA, 2009, pp. 209–216.

[8] Y. Choi, J.-S. Yang, and J. Jeong, "Application framework for multi platform mobile application software development," in *11th International Conference on Advanced Communication Technology*. ICACT, 2009, pp. 208–213.

[9] PhoneGap. (2010, May) Cross platform mobile framework. [Online]. Available: http://phonegap.com/

[10] MoSync. (2010, May) the open source standard tool for cross-platform mobile applications. [Online]. Available: http://www.mosync.com/

[11] H. Millner, P. Gulden, M. Weyn, M. Fabinder, and A. Casaca, "Description of the locon protocols. deliverable 4.2 of the fp7 locon project," Symeo, CIT, CEA-LETI, INOV, Artesis, Fraunhofer IIS, Tech. Rep., 2009.

[12] A. Chambron, M. Fabinder, N. Hadaschik, S. Wibowo, P. Gulden, K. Willame, and G. Pestana, "Concept of an inter-operable interface. deliverable 4.3 of the fp7 locon project," ANA, perLocus, Symeo, CEA/LETI, INOV, CIT, Tech. Rep., 2009.

[13] D. Pauwels, K. Fierens, G. Ergeerts, and M. Weyn, "Remote Athlete Monitoring Application for LocON," 2010.

[14] Zephyr. (2010, May) Zephyr hxm — heart rate and accelerometer data analysis. [Online]. Available: http://www.zephyr-technology.com/hxm.html

[15] M. Weyn and M. Klepal, "Adaptive Motion Model for a Smart Phone Based Opportunistic Localization System," *Mobile Entity Localization and Tracking in GPS-less Environnments*, pp. 50–65, 2009.

[16] Y. Budts, G. Ergeerts, and M. Weyn, "Internal Social Networking Application using LocON," 2010.

[17] Q. Labs. (2010, May) Qt mobility project. [Online]. Available: http://labs.trolltech.com/page/Projects/QtMobility

[18] B. Pauwels, D. Lacko, D. Vermeiren, G. Ergeerts, M. Weyn, and R. Steurs, "Smart Environment: Indoor Localization and Evaluation (SEnILE)," 2010.

 ISBN: 978-1-61208-100-7