INF-3981

Master's Thesis in Computer Science

Collecting and Distributing Sensor Data using the Argos middleware platform

Mats Mortensen

June, 2007

Faculty of Science
Department of Computer Science
University of Tromsø

Abstract

Applications that adapt to environmental and situational changes are difficult to build because computers cannot capture, represent or process context information as easily as human beings. Nevertheless, context information is very valuable because it allows applications to be made more user-friendly, flexible, and adaptable. This realization has spawned a multitude of research efforts to simplify development of context-sensitive applications. A result of one of these research efforts is the Argos middleware platform, which is an application server created specifically for personal applications that can adapt to changes in their environment.

Applications that rely on context information must often collect this information from external measurement devices, commonly known as *sensors*. These devices respond directly to physical stimulus to produce meaningful information about their surroundings. Typical examples are sensors that produce location, temperature or motion information, but they can also, for instance, be devices that monitors the physical condition of a person. The goal of this thesis has been to design, develop and evaluate functionality for the Argos middleware platform that makes it easier for Argos applications to collect and use sensor measurements.

The functionality has been developed in collaboration with the National Center for Telemedicine (NST) who wants to use Argos for monitoring patients. They intend to develop a system that can give patients semi-automatic feedback and advice on how to maintain and improve their lifestyle. To do this they want to use personal sensors that monitor attributes relevant to a patient's condition. The functionality developed in this thesis has provided a starting point for NST to develop their system and contributed lots of technical information that will prove useful for their project.

Acknowledgments

This thesis marks the end of 5 years of studying computer science at the University of Tromsø. A fantastic university in every aspect, but especially nice because of the atmosphere created by the students that study here. I honestly do not think I would have completed my education if it had not been for my fellow student friends who not only have provided indispensable technical discussions and support, but also contributed with much needed non-computer related distractions.

I would like to thank my supervisors, Anders Andersen and Arne Munch-Ellingsen, for their encouragement, ideas and support when writing this thesis. Last but not least, I want to give a thanks to my family for putting up with me through periods of great stress and give a special thanks to my brother for giving me a ride from the university on numerous occasions this semester.

Contents

Acronyms

APMS A Personal Middleware System

RUP Rational Unified Process

NST National Center for Telemedicine

COLD Chronic Obstructive Lung Disease

RDF Resource Description Language

XML eXtensive Markup Language

WSN Wireless Sensor Network

SCT Sensor Configuration Tool

ASS Argos Sensor Service

MSF Mobile Sensor Framework

DBMS Database Management System

JEE Java Enterprise Edition

SOA Service Oriented Architecture

ORM Object-Relational Mapping

HBM Hibernate Mapping File

JMX Java Management Extensions

JAXB Java Architecture for XML Binding

WM5 Windows Mobile 5.0

NMEA National Marine Electronics Association

GGA Global Positioning System Fix Data

List of Figures

Chapter 1

Introduction

1.1 Background

One of greatest challenges for future computing is to create applications that are able to detect and respond to changes in their environments. Such changes are often related to the context of given situations, but could also be simple condition changes that should trigger an application response. Humans constantly make use of situational information by deducing and interpreting information available to us from our different senses. This information is then used to generate the appropriately response or reaction to any given situation. Computers, however, do not have general way of collecting such information. They must instead rely on specialized measurement equipment that can respond directly to physical stimulus.

Tiny sensor devices that responds to heat, light, sound, pressure, motion, flow, etc are rapidly making inroads into automotive, medical, industrial, and aerospace applications in addition to a multitude of consumer electronics. The motivation behind the sensor usage can be traced to rising concerns for safety, convenience, entertainment, and effiency factors [3]. In many cases the sensor readings are only useful for embedded devices performing pre-defined tasks, but there are also many cases where such readings would be useful for cellphones or general purpose computers. A cellphone application could, for instance, benefit from current location and activity information. This information could be collected using a GPS sensor attached to a cellphone while a remote server could provide an activity service based on GPS coordinates. If a person with such a cellphone visited the cinema he could automatically be presented with information about movies, schedules, ticket purchase and related material.

However, integrating sensors with cellphones and passing sensor data on to remote sources is not a trivial task. Most types of sensors have their own proprietary data format that must be interpreted and there exists a number of different hardware protocols to communicate with them. These problems are usually dealt with every time a sensor application is developed, which increases the development time and limits code reuse.

The scenario with the cellphone and GPS sensor demonstrates that in many cases sensors will be directly connected to handheld devices that pass sensor data on to remote server-side applications. Server-side applications will just be interested in getting the sensor data and does not want to worry about sensor communication protocols or data formats. Similarly, there might be applications on a handheld device that are just interested in sensor data and not how to actually retrieve it from a sensor. These two scenarios forms the basis for what this thesis will attempt to solve.

1.2 Problem Definition

The primary goal for this thesis is to develop a system to simplify development of sensor applications on the Argos middleware platform[10]. It is assumed that such applications either run on a computer directly connected to a sensor or on a computer that receives sensor data from a remote resource. In the latter case it is assumed that the remote resource is a handheld device that is directly connected to one or more sensors.

This thesis will mainly focus on the last scenario and the system will be developed to extend the functionality of the Argos middleware platform.

The primary goals for this thesis can be summarized as follows:

- Create a sensor framework for handheld devices that takes care of accessing, retrieving and distributing sensor measurements.
- Create a sensor framework for the Argos middleware platform that can receive sensor data from handheld devices and offer it to Argos applications.

1.3 Interpretation

To simplify development of the server-side applications different types of middleware platforms are often used. Middleware or application servers can provide easy to use communication-, persistence-, and data abstractions that takes away much of the complexity regular applications are faced with. Creating a similar

abstraction for accessing sensors can make it easier both to receive sensor data from external sources and providing this data to applications running on the middleware. This thesis will attempt to develop a sensor framework for the Argos[1] middleware platform. Argos is a very lightweight application server that is primarily created for context-sensitive applications that want to collect measurement data from remote locations. The sensor framework developed in this thesis will extend this notion by giving applications a simpler way to retrieve, use and possibly store sensor data.

It is considered unlikely that sensors will be connected directly to the computer running Argos. This assumption simplifies the Argos sensor framework because it will not need to worry about providing abstractions for directly accessing sensors. However, if it does not get the data from a sensor directly it will get it from another computer or a handheld device. This thesis will, for reasons described later, assume that only handheld devices will be directly connected to sensors. The system to be developed must somehow interact with these devices to collect sensor data and provide this information to Argos applications.

The complete system will hopefully provide new perspectives on how to create software abstractions for accessing sensor data but will also investigate the benefits and potential drawbacks of the newly redesigned Argos. Among its new features is a micro architecture that allows developers to create system functionality that is completely independent from the core features of the middleware. The system developed in this thesis will use this architecture to extend Argos with a sensor framework Argos applications can use to receive sensor information.

The software developed in this thesis is not intended to be production quality, but rather a prototype that can give new perspectives on how to simplify access to sensor systems. Although it is strictly research, it is conceivable that some of the functionality developed will be used and modified to be used for other projects. For this reason it is important that the functionality developed is as generic and component based as possible.

1.4 Method and Approach

This thesis will attempt to follow an agile software development pattern called *Scrum* as described in [12]. Agile software development evolved in the mid 90s as a reaction to established methodologies like the Waterfall Model[2] and Rational Unified Process (RUP)[3]. These methodologies were criticized for being too

[1] In version 2 of middleware previously known as "APMS" the name was changed to "Argos".
[2] http://en.wikipedia.org/wiki/Waterfall_model
[3] http://en.wikipedia.org/wiki/Rational_Unified_Process

bureaucratic because they comprise of very strict project guidelines and often produce a lot of documentation. Agile methods, like Scrum, on the other hand, have a more people oriented approach and emphasize real-time, face-to-face communication over written documents.

The Waterfall Model, RUP and similar methods are development processes that all follow a workflow that generally involves the following phases:

- Analysis
- Design
- Implementation
- Testing

RUP distinguishes itself from the Waterfall Model by being iterative. This means that the workflow involving analysis, design, implementation and testing is applied multiple times before a project is completed. The Waterfall Model, on the other hand, is sequential and the workflow is only applied once. RUP and even the Waterfall Model have produced many successful applications, but they often fail to factor in that system requirements can change quickly in response to customer needs. Scrum and other agile methods have realized this and tries to welcome and adapt to change by being more "lightweight" and less dependent on documentation.

Agile development with Scrum is team based and evolves generally around two different backlogs. The product backlog contains a prioritized list of all high level product requirements and entries are added as the project proceeds. Practically anyone can add entries to the product backlog, but it typically comes from the users or stakeholders of the system. Scrum teams work in iterations called *Sprints* that typically last 2-4 weeks depending on the size of the project. They acquire a number of tasks from the product backlog that forms the sprint backlog. The sprint backlog will contain the tasks from the product backlog that they think they will be able to finish within the sprint timeframe.

The Scrum teams are managed by a Scrum Master that enforces the Scrum development rules and helps the teams make project decisions. Before a sprint is initiated, the scrum team and the Scrum Master have a sprint meeting where the master helps the team decide which tasks should be completed for the sprint. During a sprint the Scrum Master participates in daily meetings called *Daily Scrums* with the team where each team member reports their progress. This meeting is intended to discover and solve small development obstacles that impedes the

sprint progress. When the sprint is completed the team has another meeting with their Scrum Master called a *Sprint Review* where the team presents the work they have accomplished. This meeting can include other developers, management people and stakeholders, but it is coordinated by the Scrum Master. The project reaches its final stage, *Closure*, when there are no more requirements or tasks left in the product backlog. When *Closure* has been reached the final preparations for product release are started and no more features are to be added to the system.

Although this thesis will be developed according to the Scrum principles, the development process will be somewhat altered to accommodate that this is an individual project. Agile development with Scrum is really designed for larger project with teams of people working together, but the development philosophy will probably also prove advantageous for this thesis.

This thesis will have the following Scrum development details:

- **Scrum Master:** Arne Munch-Ellingsen (Cosupervisor)
- **Sprint timeframe:** 2 weeks
- **Daily Scrums:** Every Wednesday and Friday

1.5 National Center for Telemedicine (NST)

The National Center for Telemedicine (NST)[4] are working on a project to help certain patients improve their health by giving them lifestyle advice. It is a fact that many chronic illnesses can be ameliorated, postponed or avoided by just minor changes to patients day-to-day behavior. Typical examples of such diseases are Type II Diabetes, Chronic Obstructive Lung Disease (COLD) and obesity. However, NST realize that it is notoriously difficult for people to maintain permanent lifestyle changes. As pointed out in [23] lifestyle changes cannot be sustained by simple 20 minute consultations. They need to be continuously promoted, which is a very time consuming and difficult process.

In order to alleviate this problem, NST wants to develop a system that can give patients semi-automatic feedback and advice on how to maintain and improve their lifestyle. Personal sensors that monitor attributes relevant to a patient's condition can be used to predict a patient's health status and to formulate advice on how a patient can maintain a healthier way of living. This way patients can receive constant information about their health status in addition to suggestions on how they can improve their life.

[4] http://www.telemed.no/

1.6 NST's relation to this thesis

NST wants to use the Argos middleware platform to develop a prototype application that gathers sensor data from cell phones. They envision that in the future a patient can be equipped with a cell phone that interacts with various types of sensors related to the patient's lifestyle. The purpose of this prototype is to illuminate problems related to sensors and collection of sensor data.

This thesis will make two important assumptions based on the NST application usage scenario that will greatly influence the developed system. It will be assumed that:

1. In most cases the sensor will be directly connected to a handheld device that will pass data on to Argos
2. In most cases both Argos and the handheld device will be controlled by the same actor(s).

1.7 Outline

The thesis consists of the following chapters:

Chapter 1 - Introduction

Chapter 2 - Related Work
This chapter presents research and development on communicating with sensors and describing sensor data formats. It also contains a brief overview of limitations related to using mobile devices.

Chapter 3 - Requirements
This chapter presents the requirements for the system that has been developed as part of this thesis.

Chapter 4 - Technology
This chapter presents a number of different technologies that have had an impact on the developed system.

Chapter 5 - Design
This chapter presents the design of the system and its related components.

Chapter 6 - Implementation
This chapter describes how the system was implemented, which technologies were used and other technical considerations.

Chapter 7 - Experimentation
This chapter presents a demonstration application that uses the system and provides details on how the system should be used.

Chapter 8 - Evaluation
This chapter gives an evaluation of the system based on the requirements specified. It also presents a summary of its advantages and limitations and suggests how it can be improved in future work.

Chapter 9 - Conclusion
This chapter concludes the thesis and the achievements made based on the problem definition given in chapter 1.

Chapter 2

Related Work

This chapter will present research and existing systems that are relevant for this thesis. Much of this previous work focus on how to describe sensor systems and sensor data (or just data) in interoperable formats that allow easy sharing, exchange and processing of this type of information. The results of these research efforts will be presented and discussed in the light of the goals set by this thesis.

2.1 Background

Sensor systems can generally be regarded as information resources that produce some kind of data. A sensor is usually thought of as a simple device that responds to stimulus, such as heat, light or pressure, while sensor systems can comprise of multiple sensors. An example of the latter could be the Davis Weather Station II[1], which is a measurement instrument that is comprised of a multitude of sensors that measure temperature, wind speed and direction, barometric pressure, humidity and other weather related information.

Measurement instruments and sensors are usually resource constrained devices that have limited connectivity making them dependent on base stations close by to deliver their data. The base stations can be online computers or other resource rich devices that can publish sensor data on independent web sites, store them in sensor stores[19] or send them to a centralized place for processing.

In order to collect data from a sensor a computer or device must communicate with it in a certain way. The typical wireless communication technologies used

[1] http://www.davisnet.com/weather/products/weather_product.asp?pnum=07440CS

are Infrared[2], Bluetooth[3] and Wireless LAN, but a number of other technologies like ZigBee[4], Wibree[5] or Wireless USB are possible alternatives. Especially one of the flavors (Certified Wireless USB[6], WirelessUSB[7]) of wireless USB is probably going to be popular in the future. Each of these communication technologies have their own strengths and weaknesses, making it possible for sensor manufactureres to choose the one most appropriate for their sensor.

However, one of the problems that makes sensors complicated to work with is that these communication technologies can use a number of different data exchange protocols. For instance, a device that supports WLAN connectivity can use TCP/IP or UDP to transfer its sensor data while a bluetooth or infrared device may use emulated rs-232[8] serial ports or maybe OBEX[9]. A complete overview of the bluetooth protocols is given by figure 2.1. It shows the whole bluetooth protocol stack where the RFCOMM layer provides the rs-232 serial port emulation.

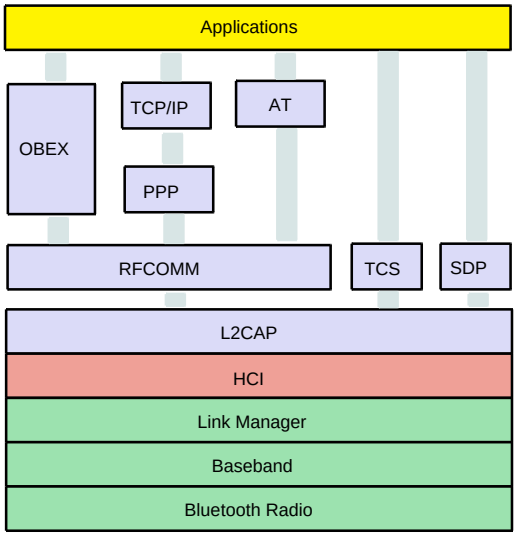

Figure 2.1: The Bluetooth protocol stack

Such a large number of different communication technologies combined with different data exchange protocols makes it difficult to create a framework that can

[2]http://www.irda.org/
[3]http://www.bluetooth.com
[4]http://www.zigbee.org
[5]http://www.wibree.com/
[6]http://www.usb.org/developers/wusb/
[7]http://www.cypress.com/portal/server.pt?space=CommunityPage&control=SetCommunity&CommunityID=209&PageII
[8]http://www.camiresearch.com/Data_Com_Basics/RS232_standard.html
[9]http://dev.zuckschwerdt.org/openobex/

be used to provide uniform access any sensor. The strategy that probably will be adopted by this thesis is to try to support a subset of the most commonly used protocols and technologies.

2.2 Bridging the sensor gap

One of the primary goals for this thesis is to bridge the gap between sensors and applications that want to use the data they provide. The lack of sensor data and communication standards makes it difficult to develop applications that want to use sensor data, but does not want the complications concerned with sensor interaction. The main problem with sensor interaction is that sensor devices are produced by a large number of different companies that usually define their own protocols and data formats. That is, they design data protocols that work on top of protocols like tcp/ip or rs-232. The use of proprietary formats forces developers to reinvent the wheel more than necessary and severly limits general purpose sensor tools from being developed.

One way this situation could be improved is to find standard ways to describe how to communicate with sensors and standard ways to describe sensor data formats. [11] suggests a way to do this by defining a generic XML format that describes both the hardware protocol used to communicate with a sensor and the protocol structure of the sensor data. An example of this format can be seen in listing 2.1.

```
<input id="xbow␣adxl202␣evaluation␣board">
   <rs232 port="/dev/ttyS0" baudrate="38400" buffsize="4"
         databits="8" stopbits="1" parity="no">
      <poll wait="1500" command="G"/>
      <packet>
         <channel id="0" name="AccX" bits="16" sign="unsigned"
            format="integer"/>
         <channel id="1" name="AccY" bits="16" sign="unsigned"
            format="integer"/>
      </packet>
   </rs232>
   <inputcolumn id="0" channel="0" name="AccX" bits="14" sign="unsigned"
      format="integer"/>
   <inputcolumn id="1" channel="1" name="AccY" bits="14" sign="unsigned"
      format="integer"/>
</input>
```

Listing 2.1: CommonSense ToolKit XML data format

The XML format illustrated by listing 2.1 contains three different sections that describes the communication protocol used (rs232), the packet format of the sensor data (packet) and a list of inputcolumns that describes data that should be

retained for further processing. The authors of this format also created a graphical tool to simplify the process of creating such sensor profiles.

The motivation for the authors of this sensor format was to find a way to deal with sensors in a uniform way. The format was developed as part of the CommonSense ToolKit[10] that aims to assist in the communication, abstraction, visualisation, and processing of sensor data. However, even though the paper fails to mention it, they seem to make some strong assumptions about sensor communication. The XML examples they provide appear to assume that a sensor will only send data packets in a binary data format and fails to describe any further sensor protocol interaction. More complex sensor systems might require additional interaction in order to setup connections, starting/stopping data transfers and dealing with partial packet receivals. An example of such a sensor system is the CoaguChek XS described in [1] used by NST. It measures INR values indicating the intensity of anticoagulation in patients blood. The measured values are used for dosage adjustments of Warfarin, which is a drug that prevents the blood from clotting. The CoaguChek XS employs a complex sensor protocol that has:

- Special control characters for initiating and terminating connections
- Different modes/states for data exchange and command exchange
- Data that is split into data frames that may be further split into multiple data blocks
- Data frames that must be acknowledged/deacknowledged with special response characters.

The protocol used by the CoaguCheck XS would be impossible to describe in the format defined by the creators of the CommonSense ToolKit. Unless the developers of the CommonSense ToolKit somehow feed their applications additional information on how to communicate with sensors the XML format they propose will only work for a very limited number of sensors.

2.3 Describing sensors and sensor data

The work done by the developers of the CommonSense ToolKit focused on defining an XML format to describe how to communicate and interpret data from specific types of sensors, but it is also interesting how sensor data can be modeled after its collected. There exists a few different technologies that can be suitable for this task even though it is not certain that it is possible or desireable to create a format that can describe any type of sensor data.

[10] http://cstk.sourceforge.net/

2.3.1 XML

One way to describe collected sensor data is by creating a generic XML format that any type of sensor data can be described in. The advantage of this approach is that XML is an interoperable format that is supported nearly everywhere allowing sensor data to easily be shared across different systems and programming languages. Furthermore, since XML is so popular there exists an abundance of tools to manipulate XML structured information. This is especially useful when a format is described by an XML schema since many programming languages have tools that can map XML to objects in a programming language automatically.

There are also a number of reasons why XML might not be suitable for describing sensor data. The native sensor data formats are usually very compact in order to use up as little space as possible, but an XML format will necessarily be very verbose (because of the tags). If the sensor data is collected and used in a resource constrained environment, like on a handheld device, the extra space overhead might matter. As described by [7] using XML instead of an equivalent binary representation can often cause up to sixteen- to seventeen-fold increase in size. While [15] suggests that the average is around 400% larger. This, of course, depends on factors like length of the tags and the structure of the document. XML also demands more processing power to be interpreted than a compact proprietary sensor data format. This might also be an issue on a device with limited resources.

Finally, XML lacks good ways to describe relationships and semantics which might limit its usefulness when compared to ontology languages.

2.3.2 The Resource Description Language (RDF)

The Resource Description Language (RDF) is a ontology language that leverages a simple and flexible data model to describe objects and their properties as described in [16]. RDF can be thought of as a way to decompose any type of knowledge into small, structured pieces of data that can include semantic information about those pieces. Since RDF data is well structured (actually a graph) it is easy for computers to reason about or query the data. Originally RDF was based on XML, but can now be expressed in a number of different syntaxes.

RDF was created to describe any type of information and might therefore be a good choice for describing sensor data. Since it can be expressed in XML it

retains most of XML's positive and negative sides, but is more suitable for describing data relationships, semantics and makes it easier to make queries about the data. However, RDF would not provide a generic way to describe sensor data. It is an ontology language and consequently is designed for describing ontologies. An ontology is basically a specifically designed vocabulary for describing domain specific information. In the case of sensors, you might, for instance, create a specific vocabulary for each type of sensor and use that vocabulary to describe the sensor data for that sensor type.

The real power of RDF can be discovered by using multiple vocabularies to describe information because it enables computers to *reason* about that information. A good example might be a vocabulary that defines the units celcius and fahrenheit typically used by temperature sensors. It is important to realize that these units are not exclusive to temperature sensors, but might show up in tons of situations regarding temperature (like weather reports). In this example it would be a good idea to define a standard vocabulary that provided definitions (using URIs) for celcius and fahrenheit. If data regarding temperature from two different, independent sensors was described in RDF using this vocabulary a computer program could "know" that the two sources were referring to the same type of data. Or if one was using celcius and the other fahrenheit it would know the difference and could convert one to the other before any comparison was made! These types of standard vocabularies can be created for ANY type of domain and by sharing vocabularies computers can suddenly start to make sense of unrelated data in ways that were previously not possible.

Even though RDF seems like a promising language for describing information it is still in its infancy and only one piece of the puzzle. It would involve a lot of work to define ontologies for every sensor type and in order for RDF to really become useful, standard vocabularies needs to be defined for various types of information (like temperature). Unfortunately not many such standard vocabularies have been defined although some are under development[11]. For this reason RDF has not been widely adopted (yet!) to describe information.

2.4 Sensor Description Languages

Sensor description languages are languages that try to describe and capture information regarding the different aspects of sensor systems. In the context of this thesis this work is interesting because we want to investigate ways to describe sensors and sensor data.

[11] http://www.w3.org/2003/01/geo/

2.4.1 SensorML

The biggest research effort in this area has been done by The Open Geospatical Consortium (OpenGIS), which is developing a standard called SensorML[6]. On their website OpenGIS have specified the following goals for SensorML.

- Discovery of sensor, sensor systems, and processes
- On-demand processing of Observations
- Lineage of Observations
- Support for tasking, observation, and alert services
- Plug-N-Play, auto-configuring, and automous sensor networks
- Archiving of Sensor Parameters

However, SensorML is currently nothing more than an unfinished XML based specification that provides a generic data model to describe sensors and sensor systems. It uses GML[2], a language also created by OpenGIS, to describe the geospatical properties of sensors. A sensorML description of a sensor system can contain the following:

- Identification information like manufacturer, name, model, etc
- What specific sensors (called *detectors*) the system is made up of
- What physical entities the system measures
- What accuracy it can achieve
- Other types of metadata that describe its capabilities or characteristics

Additionally, the specification allows a sensor system description to contain process chains that can transform measurements into more useful information. Typically a process chain could apply a well known formula (like the formula for calculating the wind-chill factor[12]) and thus produce another useful measurement.

A tutorial on how to describe the Davis Weather Station is provided by the SensorML website[13]. Figure 2.2 illustrates what sensor components the weather station is made up of and what type of data each sensor provides.

[12] http://en.wikipedia.org/wiki/Wind_chill

[13] http://vast.uah.edu/joomla/index.php?option=com_content&task=view&id=25&Itemid=50

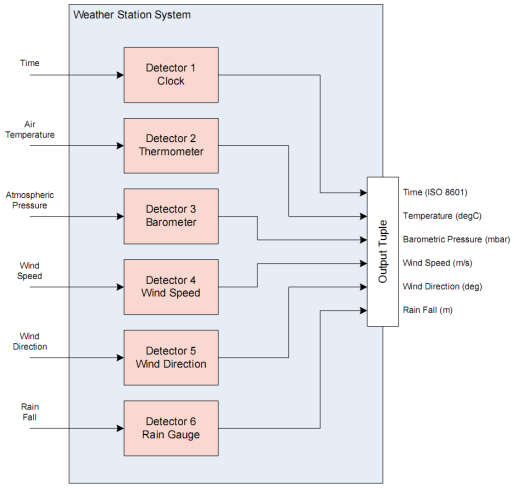

Figure 2.2: Davis Weather Station

The system as a whole is described by a comprehensive metadata section that includes identification, classification and capabilities of the system. The identification part for the Davis Weather Station can be seen in listing 2.2.

```
<identification>
   <IdentifierList>
      <identifier name="longName">
         <Term qualifier="urn:ogc:def:identifier:longName">
            Davis Weather Monitor II Station
         </Term>
      </identifier>
      <identifier name="shortName">
         <Term qualifier="urn:ogc:def:identifier:shortName">
            Davis Weather Station
         </Term>
      </identifier>
      <identifier name="modelNumber">
         <Term qualifier="urn:ogc:def:identifier:modelNumber">
            7440
         </Term>
      </identifier>
      <identifier name="manufacturer">
         <Term qualifier="urn:ogc:def:identifier:manufacturer">
            Davis Instruments
         </Term>
      </identifier>
   </IdentifierList>
</identification>
```

Listing 2.2: Identification metadata

Similar identification metadata is provided for each detector, but is less detailed. Each detector also has its own input/output section that can describe one scalar

input and one scalar output. Usually detectors only measure a phenomena and do not need any input in which case the input is modeled as identical to the output. The input/output metadata for the temperature sensor that is part of the Davis Weather Station is illustrated in figure 2.3.

As can be seen from figure 2.3 each input/output is described by a scalar parameter element (<quantity>) that uses URIs to define the type and unit of the data. SensorML attempts to provide a library of URIs to define common phenomenon data types and units. This ensures that anyone using SensorML will use and expect the same type of input/output values. Unfortunately it does not seem to exist any documentation on this URI library, except from what can be found in the XML schemas and examples.

```
<inputs>
   <InputList>
      <input name="temperature">
         <swe:Quantity definition="urn:ogc:def:phenomenon:temperature"
            uom="urn:ogc:def:unit:celsius"/>
      </input>
   </InputList>
</inputs>

<outputs>
   <OutputList>
      <output name="measuredTemperature">
         <swe:Quantity definition="urn:ogc:def:phenomenon:temperature"
            uom="urn:ogc:def:unit:celsius"/>
      </output>
   </OutputList>
</outputs>
```

Listing 2.3: Detector input/output metadata

Although SensorML provides a good and rich data model for expressing sensor meta information it has a few limitations. First of all the specification is fairly complex and consists of thirteen different XML schemas that define the various elements it is made up of. Secondly, there are very few, if any, tools that can assist in using SensorML. Combined with scarce documentation that consists of little more than one tutorial and a few demonstration XML documents it would be difficult and time consuming to base a project upon it. This problem is likely to disappear when the product becomes more mature, but currently this makes SensorML unsuitable for most projects.

Another limitation of SensorML is the choice of XML as its foundation. While XML has a well defined syntax, it lacks a good way to describe semantics and relationships. According to [9] this will limit the potential for reuse and interoper-

ability of SensorML profiles since the profiles cannot share a common vocabulary. It is puzzling why OpenGIS did not decide to use a ontology language like RDF or OWL[14] instead of XML, but research efforts like [9] have realized this limitation and done research on building ontologies on top of SensorML.

The final limitation of SensorML is that it does not define any standard format for the actual sensor data. Perhaps this never was their intention, but according to this paper[19] this is one of their future goals.

SensorML is not released yet for general use. The specification available for download is a pre-release that according to their website is subject to change.

2.4.2 TinyML

TinyML[18] is another sensor description language that focuses exclusively on embedded wireless sensor networks (WSN)[8]. It uses many of the same ideas as SensorML, but is much more lightweight and was designed to overcome some of SensorML's deficiencies.

Originally the creators of TinyML realized that in the future we will access and exchange data from multiple sensor networks that are not necessarily compatible with each other. This realization resulted in an XML based language that provides a universal interface for interacting with sensor networks. TinyML explores two-way interaction between users and sensor networks as well as interaction between the sensor networks themselves.

The basic components of TinyML are sensors, platforms and sensor fields. The platform represents a physical device that has a processor, energy source and radio capability. Each platform typically has a collection of sensors where each sensor describes a specific sensor and its properties. A sensor network is described by a sensor field, which is a collection of platforms.

While TinyML employs some nice concepts it is primarily created for sensor networks, which makes it difficult to adapt to other sensor systems. Furthermore, it does not have any examples or implementations available, but provides useful research information regarding sensor systems.

2.5 Limitations of Handheld devices

The problem description in chapter 1 assumes that in most cases sensors will be directly connected to handheld devices like cellphones. Since people usually keep

[14] http://www.w3.org/TR/owl-features/

their cellphones close it is also likely that the cellphones will be in close proximity to personal sensors, for instance sensors that monitors patients. This is how NST intends to use sensors and like mentioned in 1.6 they also want an easy way for sensor data collected by handheld devices to be transmitted to Argos applications.

This section will describe some limitations regarding cellphones that may affect how sensor data collection should be performed.

2.5.1 Limitations

Typically when you develop applications for handheld devices you have to take into consideration that such devices have limited resources and features. Some of these limitations, as described by [21] and [24], include:

- Limited computing power (such as processor speed, memory size and disk capacity)
- Short battery life
- Low-bandwidth connectivity and reliability
- Limited display features

Our area of concern involves the collection and distribution of sensor data in which case the most important limitations of handheld devices will be the limited connectivity and computing power available. Potentially a cellphone can collect sensor data from multiple sensors and processing this information may require a lot of computing power, especially if the collection frequency is high. Furthermore, if this information should be transmitted to a remote server the bandwidth of a GPRS[13] (160 kbit/s) or even EDGE[13] (384 kbit/s) internet connection might be a limiting factor. The cost of sending the data might also be an issue because many internet providers charge mobile devices by the byte. For instance, the norwegian telecommunications company Telenor[15] charge users kr 20,- pr. MB up to a maximum of 50 kr a day[16] for using their mobile wireless network.

These issues must be taken into consideration when deciding how a mobile device should represent and distribute sensor data. The representation may end up being a choice between the efficiency of a proprietary protocol and the decoupledness of an XML type protocol. The disadvantage of XML in this case is the cost of parsing and the extra size it will add to the sensor data as discussed in

[15] http://www.telenor.no
[16] http://telenormobil.no/priser/tjenester/gprs/

2.3.1. One way of reducing the overhead caused by XML is by using compression techniques as described in [7], but this has the drawback of using more computing resources on the device.

In the NST usage scenario limitations like connectivity and computing power are of less importance because sensors monitoring patients will generate small amounts of data that do not need to be sent frequently. However, the system developed in this thesis will try to be as general as possible and consequently all limitations are taken into consideration.

2.5.2 Distributing sensor data

In our problem scenario it must be possible for handheld devices to distribute collected sensor data to remote receivers. This can be accomplished by either using push-based approach where the device sends data to a receiver at regular intervals or using a pull-based approach where the server polls the device for data when it needs it.

Pull-based approach

Using a pull-based approach is appealing because receiving sensor applications might not need sensor data all the time, but only in certain situations. This approach can thus greatly reduce the communication to applications wanting to receive sensor data and to the sensors themselves. Less communication would also mean reduced processing and increased battery life for the device. However, using a pull-based approach for communicating data is difficult with handheld devices. The main problem is that mobile devices are not always connected to the internet and when they are they do not have a static IP as described in [4]. This greatly reduces the usefulness of a pull-based approach because the only way to poll the device would be by using SMS messages. SMS is not only very slow, but is also expensive from a financial point of view.

Push-based approach

A push-based approach is more desireable if the extra communication is not an issue. This is because the only assumptions that have to be made is that:

i The device knows the address of the receiver

ii The receiver offers a mechanism for receiving data

The best solution will depend on a sensor application's usage pattern. If the application only needs sensor data occasionally and sporadically it would be best to pull the data otherwise if the pattern is more regular and frequent it would be better to push the data.

2.6 Summary

This chapter has primarily focused on illuminating various aspects of describing sensors and sensor data, but has also discussed limitations and problems related to connecting handheld devices to sensors and using them to collect and distribute sensor data. It is clear that little research has been done regarding generic architectures for describing sensors and sensor data, but efforts like SensorML seems to be promising for the future.

Chapter 3

Requirements

This chapter presents the different parts of the system that needs to be designed along with the requirements they must fulfill. The requirements are based on the problem definition given in chapter 1. A more complete design of the system will be given in chapter 5.

3.1 System Goals

The main goal for this thesis is to create a system that can simplify the development of Argos applications that use sensor measurements. User applications running on Argos should be able to receive and use sensor data without having to concern themselves with how or where this data was collected. Furthermore, it is also the intention to create a system that makes it easier to develop and configure applications that access sensors directly.

It is assumed that sensors will be connected to handheld devices that in turn will pass data on to Argos. This is the scenario described by NST and in section 1.6 of the introduction chapter it was also assumed that the software on both the Argos and the handheld device is controlled by the same actors.

Based on these assumptions and the usage scenario outlined by NST the following system components will be developed:

- A tool for creating sensor configurations and sensor profiles
- A Sensor System Service for Argos
- A sensor framework for handheld devices

The first component will be a independent GUI application that interacts with the Argos Sensor Service to make it easy to create sensor profiles and sensor configurations. A sensor profile will describe a particular type of sensor and contain information on how sensor measurements can be extracted. Sensor configurations will be based on or linked to specific sensor profiles and will specify all configuration options related to the collection of sensor data from actual sensors. This information can include how to connect to sensors, how often and where to send sensor data, etc. Devices connected directly to sensors will use the sensor configurations to perform the actual sensor interaction and data extraction.

The Argos Sensor Service will be responsible for managing sensor profiles and sensor configurations created by the sensor configuration/profile tool. It will also provide ways to receive sensor data from handheld devices so that this information can be distributed to Argos applications.

Finally, a small sensor framework/library will be developed for handheld devices that are directly connected to sensors. This framework will simplify sensor interaction by using sensor configurations created by the sensor configuration/profile tool. It will also be responsible for distributing collected sensor data to the Argos Sensor Service and other potential recipients.

The requirements for this system will be split into three sections, one for each of the system components just described. For the remainder of this thesis the tool for creating sensor configurations and profiles will be referred to as the *Sensor Configuration Tool*. The sensor system service for Argos will be referred to as the *Argos Sensor Service* and the sensor framework for handheld devices will be named the *Mobile Sensor Framework*.

3.2 Functional Requirements

Functional requirements specify the technical details of a system's behavior. A common way to capture such requirements is through building *use-cases* that describe interactions between the system and external *actors* [5]. An actor is a role that is played either by a person or by some other entity. Actors may be end-users of the system, other services that interact with the system or even different parts of a large system.

Each system component is presented in its own section accompanied with the relevant use-case diagrams. With each use-case diagram follows tables that describe each individual use-case. To distinguish between the different parts of the system each of the separately developed system components will have its own ID.

The ID given is incremented for each use-case described. For instance, the Sensor Configuration Tool's first use-case is given the ID *SCT-100* in the table. To differentiate between the importance of the different requirements the keywords *could*, *should*, and *shall* are used.

3.2.1 The Sensor Configuration Tool

The sensor configuration tool is likely to have two different types of actors; Users that create sensor profiles and users that create and use sensor configurations. However, these tasks are not entirely separate and it will often be the case that the same user plays both roles.

Common to them both is that they need to connect to the Argos, which is described by use-case diagram 3.1.

Figure 3.1: Sensor Configuration Tool - Connect to Argos Use-case Diagram

Use case T-1: Connect To Argos

Requirement ID	Description
SCT-100	The Sensor Configuration Tool shall provide a way for a user to connect to a remote Argos that runs the Argos Sensor Service.
SCT-101	When successfully connected to a remote Argos, the Sensor Configuration Tool shall retrieve and display available sensor profiles that the Argos Sensor Service has stored.

Sensor profiles

For each sensor type it will be possible to create a sensor profile that describes the characteristics of that particular type of sensor. The use-case diagram 3.2 listed in this section displays the use-cases related to creating and managing sensor profiles using the Sensor Configuration Tool. The tables in this section presents the requirements for each of these use-cases.

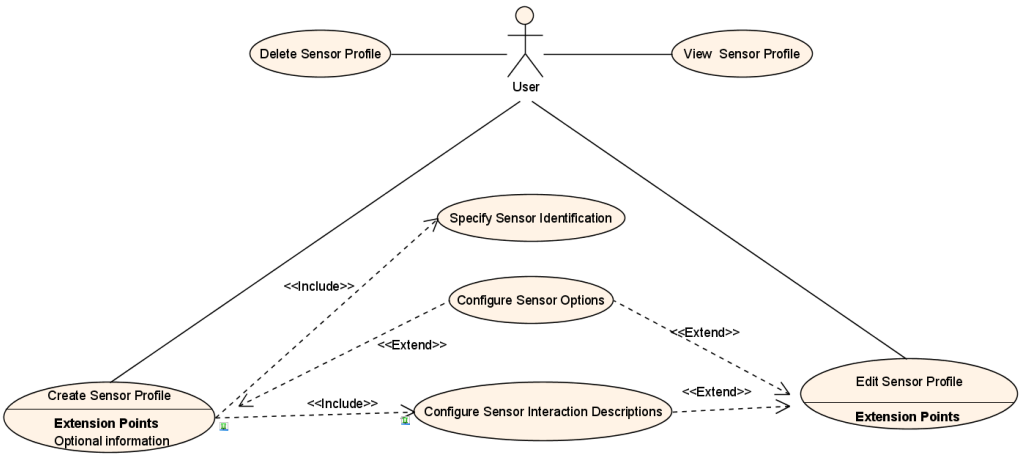

Figure 3.2: Sensor Configuration Tool - Sensor Profiles Use-case diagram

Use case T-2: Create Sensor Profile

Requirement ID	Description
SCT-102	The Sensor Configuration Tool shall provide a way for the user to create a new sensor profile.
SCT-103	A Sensor Profile shall minimally consist of: • A sensor identification as specified by use-case T-6 • A number of sensor options as specified by use-case T-7 • One or more sensor interaction descriptions as specified by use-case T-8
SCT-104	When created, a sensor profile shall be persistently stored at the remote Argos that the user is connected to.
SCT-105	The Sensor Configuration Tool shall ensure that the sensor type is unique among the already existing profiles.

Use case T-3: Edit Sensor Profile

Requirement ID	Description
SCT-106	The Sensor Configuration Tool shall allow a user to edit an existing sensor profile.
SCT-107	When a profile has been edited the remote Argos Sensor Service shall update its persistent version of the profile.

Use case T-4: Delete Sensor Profile

Requirement ID	Description
SCT-108	The Sensor Configuration Tool shall allow a user to delete an existing sensor profile.
SCT-109	After a profile has been deleted the remote Argos shall delete its persistent version of the profile.

Use case T-5: View Sensor Profile

Requirement ID	Description
SCT-110	The Sensor Configuration Tool shall allow a user to view an existing sensor profile.
SCT-111	The view shall present the user with an overview of the items the profile consists of.

Use case T-6: Specify Sensor Identification

Requirement ID	Description
SCT-112	The Sensor Configuration Tool shall allow a user to specify a set of strings that will make up the sensor identification when creating a sensor profile.
SCT-113	The set of strings shall uniquely identify a sensor profile.

Use case T-7: Configure Sensor Options

Requirement ID	Description
SCT-114	The Sensor Configuration Tool shall allow a user to specify a number of sensor options when creating a sensor profile. The options will be specific options available to the type of sensor the profile describes.

Use case T-8: Configure Sensor Interaction Descriptions

Requirement ID	Description
SCT-115	The Sensor Configuration Tool shall allow a user to add a number of sensor interaction descriptions when creating a sensor profile.
SCT-116	A sensor interaction description shall provide the information necessary to extract and interpret the measurements the sensor provides. If it is possible to describe this information in a interoperable format only one such description needs to be added.
SCT-117	The Sensor Configuration Tool shall ensure that no duplicate sensor interaction descriptions can be added.

Sensor Configurations

The use-case diagram 3.3 in this section displays the use-cases related to creating sensor configurations using the Sensor Configuration Tool. The tables in section presents the requirements for each of these use-cases.

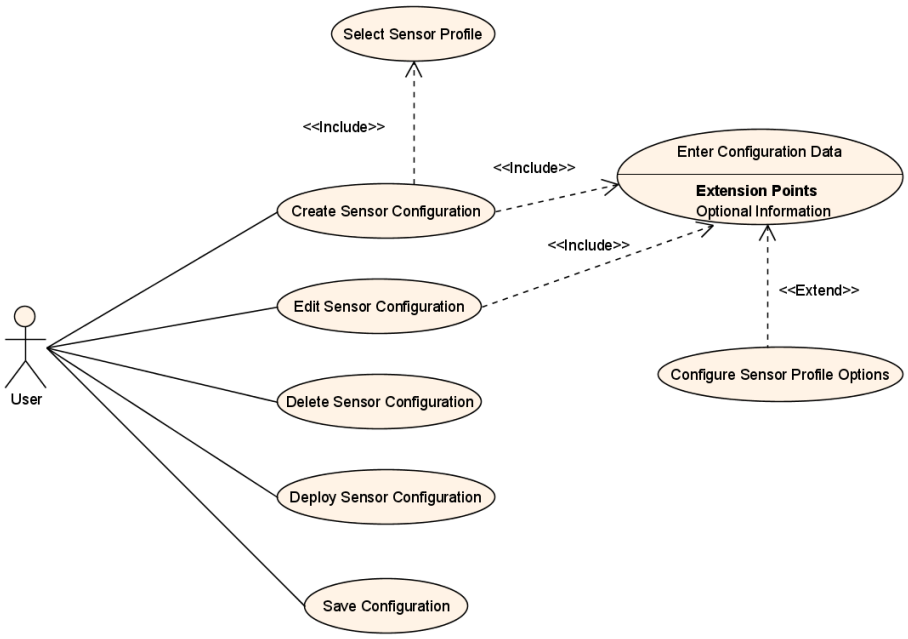

Figure 3.3: Sensor Configuration Tool - Sensor Configurations Use-case Diagram

Use case T-9: Create Sensor Configuration

Requirement ID	Description
SCT-118	The Sensor Configuration Tool shall allow users to create sensor configurations.
SCT-119	Before a user is allowed to create a new configuration he/she must select an available sensor profile to base the configuration on.
SCT-120	The configuration shall minimally consist of: • A name for the configuration. • A name for the sensor that is unique to the device connected to the sensor • A sensor profile • A start/stop status • Zero or more sensor options.

Use case T-10: Edit Sensor Configuration

Requirement ID	Description
SCT-121	The Sensor Configuration Tool shall provide a way for the user to edit an existing sensor configuration.

Use case T-11: Delete Sensor Configuration

Requirement ID	Description
SCT-122	The Sensor Configuration Tool shall provide a way for the user to delete an existing sensor configuration.

Use case T-12: Save Sensor Configuration

Requirement ID	Description
SCT-123	The Sensor Configuration Tool should provide a way for the user to save an existing sensor configuration to be used independently of the Sensor Configuration Tool.

Use case T-13: Configure Sensor Profile Options

Requirement ID	Description
SCT-124	The sensor configuration shall enable the user to configure options specific to the sensor type. These options will depend on what options are specified by the sensor profile the configuration is based on.
SCT-125	When a sensor profile is updated with new sensor specific options, these options shall become available in all sensor configurations that use this sensor profile.

Use case T-14: Deploy Configuration

Requirement ID	Description
SCT-126	The Sensor Configuration Tool shall make it possible for a user to deploy a new sensor configuration.

3.2.2 The Argos Sensor Service

The Argos Sensor Service will interact with the Sensor Configuration Tool, user applications running within Argos and with devices that provide sensor data. Consequently these systems will be the actors in the use-cases presented in this section. The main responsibilities of the Argos Sensor Service will be to:

- Store sensor profiles persistently
- Store sensor configurations persistently
- Hand out sensor configurations to the Mobile Sensor Framework
- Receive sensor data from devices running the Mobile Sensor Framework
- Distribute received sensor data to Argos applications

The requirements for the Argos Sensor Service are abbreviated ASS.

Interaction with the Sensor Configuration Tool

Use-case diagram 3.4 illustrates how the Argos Sensor Service interacts with the Sensor Configuration Tool.

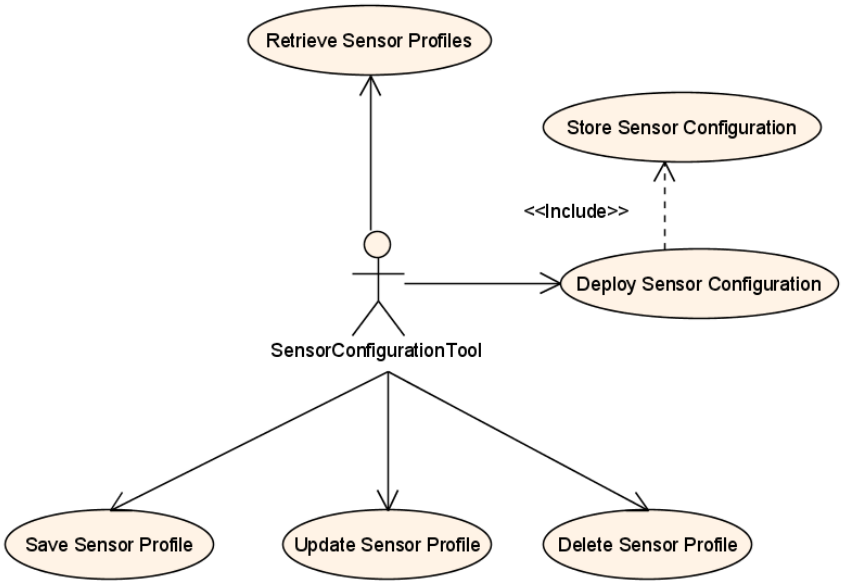

Figure 3.4: Interaction with the Sensor Configuration Tool

Use case A-1: Retrieve Sensor Profiles

Requirement ID	Description
ASS-100	The Argos Sensor Service shall hand out all stored sensor profiles when requested by the Sensor Configuration Tool

Use case A-2: Save/Update/Delete Sensor Profiles

Requirement ID	Description
ASS-101	The Argos Sensor Service shall provide a way for the Sensor Configuration Tool to save, update and delete sensor profiles.
ASS-102	The Argos Sensor Service shall check that new sensor profiles have a unique sensor identification and reject non-unique sensor profiles.

Use case A-3: Deploy Configuration

Requirement ID	Description
ASS-103	The Argos Sensor Service shall provide a way for the Sensor Configuration Tool to deploy a sensor configuration.
ASS-104	When the Argos Sensor Service receives a deployed sensor configuration it shall notify the device the configuration belongs to that a new or modified sensor configuration is available.
ASS-105	When the Argos Sensor Service receives a deployed sensor configuration it shall notify Argos applications that a new or modified sensor configuration is available.

Use case A-4: Store Sensor Configuration

Requirement ID	Description
ASS-106	The Argos Sensor Service shall store sensor configurations received from the Sensor Configuration Tool
ASS-107	The Argos Sensor Service shall generate an ID for each stored configuration that is unique within the Argos Sensor Service.

Interaction with the Mobile Sensor Framework

Use-case diagram 3.5 illustrates how the Argos Sensor Service interacts with the Mobile Sensor Framework.

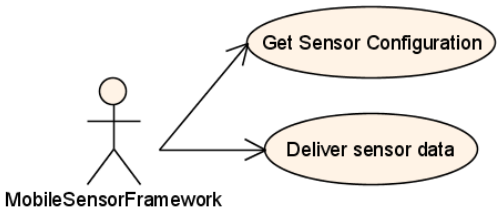

Figure 3.5: Interaction with the Mobile Sensor Framework

Use case A-5: Get Sensor Configuration

Requirement ID	Description
ASS-108	The Argos Sensor Service shall provide a way for the Mobile Sensor Framework to retrieve sensor configurations.
ASS-109	Configurations shall be identified by the unique IDs generated by the Argos Sensor Service upon their creation.

Use case A-6: Deliver Sensor Data

Requirement ID	Description
ASS-110	The Argos Sensor Service shall provide services that allow the Mobile Sensor Framework to deliver sensor data.

Interaction with Argos applications

The Argos Sensor Service will also interact with applications running within Argos. Argos applications should be able to register to receive sensor data and be able to subscribe for notifications concerning sensor configurations being deployed. The use-case diagram 3.6 illustrates these interaction scenarios.

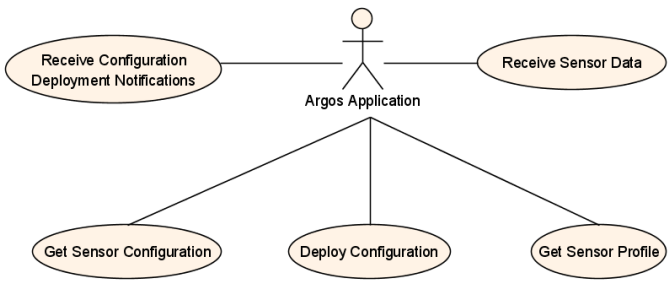

Figure 3.6: Interaction with Argos applications

Use case A-7: Receive Configuration Deployment Notifications

Requirement ID	Description
ASS-111	The Argos Sensor Service shall provide a way for Argos user applications to register to receive notifications when a new sensor configuration is deployed.
ASS-112	The notification shall at least contain the following information: • The unique ID assigned to the sensor • The name of the sensor • The sensor profile identification • The status of the sensor (If it is active/passive)

Use case A-8: Receive sensor data

Requirement ID	Description
ASS-113	The Argos Sensor Service shall provide a way for Argos user applications to register to receive sensor data from a deployed sensor. Argos applications shall supply the unique ID of the sensor configuration when they register.

Use case A-9: Deploy Sensor Configuration

Requirement ID	Description
ASS-114	The Argos Sensor Service shall provide a way for Argos user applications to deploy a sensor configuration.
ASS-115	The sensor configuration shall have the same format as the configuration produced by Sensor Configuration Tool.

Use case A-10: Get sensor configuration

Requirement ID	Description
ASS-116	The Argos Sensor Service shall provide a way for Argos applications to retrieve sensor configurations for sensors that have been deployed. To do this the applications must supply the unique ID of the configuration.

Use case A-11: Get sensor profile

Requirement ID	Description
ASS-117	The Argos Sensor Service shall provide a way for Argos applications to retrieve sensor profile information. To do this applications must supply the unique sensor profile identification.

3.2.3 The Mobile Sensor Framework (MSF)

The mobile Sensor Framework runs on a handheld device and is the framework/library used to extract measurements from sensors. The framework will interact with the Argos Sensor Service and potentially other mobile applications running on the device to get configurations and to deliver sensor data.

The use-case diagram in figure 3.7 illustrates how the Argos Sensor Service and applications that uses the Mobile Sensor Framework interacts with the Mobile Sensor Framework. The requirements for the Mobile Sensor Framework will be abbreviated MSF.

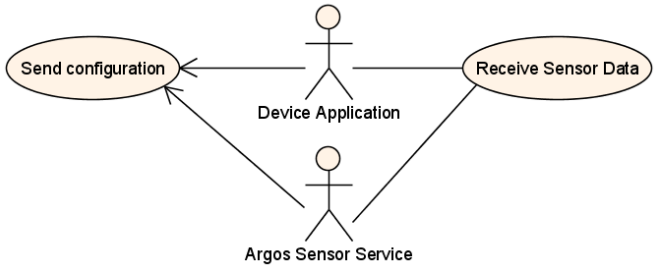

Figure 3.7: Interaction with the Argos Sensor Service and device applications

Use case M-1: Send configuration

Requirement ID	Description
MSF-100	The Mobile Sensor Framework shall be able to receive sensor configurations from the Argos Sensor Service.
MSF-101	The Mobile Sensor Framework shall be able to receive sensor configurations from applications using the Mobile Sensor Framework on handheld devices.

Use case M-2: Receive Sensor Data

Requirement ID	Description
MSF-102	The Mobile Sensor Framework shall be able to distribute sensor data to the Argos Sensor Service.
MSF-103	The Mobile Sensor Framework should be able to distribute sensor data to the applications using the Mobile Sensor Framework on the handheld device.
MSF-104	The sensor data shall be distributed in intervals specified by the sensor configuration.
MSF-105	The Mobile Sensor Framework should be able to receive sensor data from multiple sensors simultaneously

3.3 Non-Functional Requirements

Non-functional requirements comprise of requirements related to the operation, quality and constraints of a system. Stakeholders might, for instance, want certain properties or characteristics in a system that is not tied to specific functionality.

It is often useful to distinguish between non-functional requirements related to the operation of a system and non-functional requirements related to the development qualities of a system. The former can be referred to as *run-time qualities* while the latter can be labeled *development-time qualities*[14].

3.3.1 Run-time qualities

The run-time requirements of a system specifies certain *operational qualities* that the system should satisfy according to stakeholders. These *operational qualities*

can include, but are not limited to[14]:

- Usability
- Configurability
- Correctness, reliability, availability
- Quality of Service (QoS)
- Safety
- Scalability

The system presented in this thesis does not have any run-time requirements. The reason is that the run-time requirements applicable to this system have no value in a research setting and are more appropriate for production systems.

3.3.2 Development-time qualities

The development-time qualities of a system relates to architecture and design goals that will make it easier to maintain, extend and modify the system in the future. Development-time qualities can include[14]:

- Localizability (The ability to support regional differences)
- Modifiability (The ability to modify a system)
- Extensibility (The ability to extend a system with new functionality)
- Composability (Relates to the modularity of a system)
- Reuseability (The ability to reuse parts of the system)
- Interoperability (The ability to run, communicate or cooperate with different platforms)

Development-time qualities are important to the system developed in this thesis. Although it is just a prototype, achieving a high degree of extensibility and interoperability will enable the system to more easily be used in settings involving different platforms and will make it easier to maintain and develop in the future. The requirements defined for extensibility and interoperability are listed in the following two tables.

Extensibility

Requirement ID	Description
NFR-100	The Mobile Sensor Framework shall be extendible with new protocols for communicating with sensors.
NFR-101	The Mobile Sensor Framework should be designed to easily accommodate changes to the sensor configuration format.
NFR-102	The sensor configuration format shall be able to incorporate new options with minimal changes to the system components that use it.

Interoperability

Requirement ID	Description
NFR-103	The sensor configuration format shall use a standardized, interoperable format that can be interpreted by multiple platforms.
NFR-104	A generic, interoperable format shall be defined for describing sensor data.
NFR-105	The Mobile Sensor Framework shall distribute sensor data using the format from requirement NFR-104.
NFR-106	The format for describing sensor profiles shall use a standardized, interoperable format that can be interpreted by multiple platforms.
NFR-107	The Mobile Sensor Framework and Argos Sensor Service shall be interoperable and loosely coupled. This means that they should not have any interrelated dependencies and should not depend on what specific programming language or architecture the other component is implemented in.

3.4 Summary

This chapter has presented the functional and non-functional requirements of the different parts of the system. These requirements will serve as guidelines for the design and implementation of the system in addition to being the standard that the completed system is measured against.

Chapter 4

Technology

This chapter will describe different technologies that are either used by or otherwise have had an impact on this thesis. It will focus on relevant information concerning the problems these technologies are meant to solve and will briefly mention other alternatives that exist.

4.1 Overview

The problems intended to be solved in this thesis are strongly linked to interoperable technologies for describing information and tools that can simplify development. Many of the technologies presented will be linked to the Argos middleware platform, which plays a key role in the problem definition outlined in chapter 1. Consequently, the start of this chapter will provide details about Argos and similar systems.

4.2 Application servers

Software development has changed rapidly in the last few decades giving us the ability to steadily create more complicated systems. This process has been possible because the tools for creating software are gradually becoming more sophisticated. Programming languages constitute our main tool for development and it is obvious that the high level languages of today are an order of magnitude more productive than older languages. Just imagine creating one of today's huge Java systems using a programming language like Pascal[1], it would not be possible!

However, it is not just programming languages that have improved development

[1] http://pascal-central.com/

productivity. Application Servers is a relatively new software concept that aim to enable rapid application development and deployment by providing easy access to commonly used technologies and services. There exists many[2] definitions of exactly what an application server is, but the definition given by IONA[3] to describe their Orbix E2A Application Server seems to capture the most important characteristics. It defines an application server to be:

A software platform that provides the services and infrastructure required to develop and deploy middle-tier applications

Middle-tier applications perform business logic and typically reside between slim clients and back-end storage facilities. Some of the features commonly provided by application servers include web servers, DBMS solutions, support for distributed objects (DCOM, CORBA), transaction systems and various ways to do connectivity. However, the features supported varies greatly depending on the type of application server and the market segment it is created for.

Some of the advantages of using an application server include:

- **Separation of concerns**
  Makes it easier to create systems that separate business logic, presentation logic and data storage.

- **Centralized configuration** All technologies provided by an application server can usually be configured once to work for all applications.

- **Security** The application server can provide basic security layers for its services in order to reduce the security responsibility of applications.

The Java platform has had great success with its Java Enterprise Edition (JEE) application servers. Sun, the company that created Java, designed a specification called the Java Enterprise Edition that vendors follow when they implement JEE application servers. This enables different application servers that follow the JEE specification to be compatible with each other. In theory this enables applications written for one platform to run on any other JEE compliant platform[17]. WebLogic Server (BEA), JBoss (Red Hat) and WebSphere (IBM) are all commercial application servers that follow the JEE specification.

[2] http://news.com.com/2100-1001-214783.html?legacy=cnet
[3] http://www.iona.com/support/docs/e2a/asp/5.1/platform/manage/glossary.html

4.3 The Argos Middleware Platform

The Argos[4] middleware platform[10] is a personal application server that is specifically designed for context sensitive applications. It is implemented in Java, but does not follow the JEE specification. JEE compliance was never an issue because it does not have the same target audience as JEE application servers. A JEE product, is, as the name implies, targeted at the enterprise while the Argos is intended to be a small, personal and flexible application server.

Argos allows you to easily create and deploy software components and bind them together into services. A "service" created for Argos represents software that provides functionality to an end-user in the same way as a regular application would. A personal service oriented middleware like Argos makes it easier to include context awareness and reasoning in applications since support for binding persistence is part of the support offered to services deployed in the container.

The Argos application server employs a Service Oriented Architecture (SOA). Although many different definitions of SOA exist, SOA is often defined to be an architectural style whose goal is to achieve loose coupling among the interacting software components. Argos achieves such an architecture by separating its core functionality into independently managed software modules. Functionality such as service distribution, inference, transactions, persistence, and remote bindings should not be regarded as features that are needed by everyone. Instead, this functionality should be regarded as add-on services that can be deployed when necessary. This design philosophy makes each system component completely independent from the rest of the system and makes it much easier to customize Argos with only the functionality that is needed.

Out of the box Argos only provides an environment to deploy and manage services. There exists two types of services; *user services* and *system services*. Regular applications are built as user services while system services provide additional functionality to other user- or system services. Some of the system services that are currently available for Argos provide support for bindings, transactions and persistence, which makes applications that need this type of functionality faster and more reliable to develop. The beauty of this architecture is that applications can just pick the functionality they need from available system services and use them in any way they want to extend their own functionality.

[4] In version 2 of middleware previously known as "APMS" the name was changed to "Argos".

4.4 Argos System Services

Several system services have been developed for Argos that can simplify the development of user applications or other system services. In this section a selected number of services that have an impact on this thesis will be presented.

4.4.1 Hibernate System Service

Hibernate[5] is a high performance Object-Relational Mapping (ORM) solution for Java that is wrapped in a Argos system service. It provides a partial solution to the impedance mismatch problem[6] associated with storing objects in a relational database. ORM tools do not store anything themselves. Instead they map objects from a specific programming language automatically to tables in databases. Hibernate supports a number of different databases, but the only database currently supported in Argos is Apache Derby[7].

In order for Hibernate to be able to store Java objects persistently in a database the developers must describe how their objects should be persisted by using either annotations or hibernate mapping files. A hibernate mapping file (.hbm.xml) is a XML file that tells Hibernate what table and which columns it should use to store a Java object. It can also contain fairly advanced configuration options on how Hibernate should treat interconnected objects.

Hibernate is an excellent tool when everything works, but unfortunately, due to the lack of good documentation, it is difficult to get the mapping files right. Having to learn how to create mapping files is the biggest disadvantage to using Hibernate especially considering that most developers can use any SQL database without having to learn anything extra.

4.4.2 SMS System Service

The SMS service makes it possible to send and receive SMS messages if you have access to the Telenor Mobile CPA Proxy[8]. It is limited to receive messages only from phones that have a Telenor mobile subscription.

4.4.3 Web Service System Service

Argos has its own WS service that provides easy access to using and exposing Web Services using either SOAP or XML-RPC. This service is based on Apache

[5] http://www.hibernate.org/
[6] http://www.oracle.com/technology/products/ias/toplink/doc/10131/main/_html/undtl002.htm
[7] http://db.apache.org/derby/
[8] http://cpa.telenor.no/cpa/

AXIS[9] that provides a Java implementation of SOAP. Argos services can expose web methods by using a set of annotations supplied by the service.

4.4.4 The TCP/IP System Service

The Argos TCP/IP service provides a tcp/ip communication abstraction for Argos user applications and system services. It utilizes a scalable architecture based on the Java Non-Blocking IO libraries[10] in order to provide a high performance connectivity framework that can support numerous simultaneous tcp/ip connections. The service is primarily suited for Argos services that need server functionality, but it also provides tcp/ip client connectivity.

The Argos TCP/IP service provides a single threaded non-blocking IO solution that can be used by all Argos services. It allows Argos services to register tcp/ip servers on different ports and create tcp/ip client connections to remote servers. When a tcp/ip connection is established the Argos service is given a stream abstraction representing the connection. It allows the Argos service to read data (blocking), write data (non-blocking) or close the connection.

4.4.5 JMX Connectors

One of the core technologies in Argos is the Java Management Extensions (JMX)[11]. JMX provides a simple and standard way to monitor and manage Java applications both locally and remotely. A big advantage for Argos services is that Argos utilizes JMX to enable any service to expose interfaces to remote applications by using simple annotations. By using JMX connectors remote clients can connect to Argos and use the exposed methods just like they were local methods.

4.5 XML technologies

XML ended up being the technology of choice for defining many of the interoperable data formats used in this thesis. The pros/cons of xml was previously discussed in section 2.3.1. This section will focus more on specific XML technologies and the benefits they provide.

4.5.1 XML Schemas

The basic principle of XML Schemas is to define constraints and rules for how XML documents shall be structured. XML Schemas are themselves specified in XML which makes them as interoperable as the XML documents they describe.

[9] http://ws.apache.org/axis/
[10] http://java.sun.com/j2se/1.4.2/docs/guide/nio/
[11] java.sun.com/products/JavaManagement/

All XML documents, including XML schemas, must abide to the XML syntax. Proper XML documents are referred to as being *well formed*[12] if the syntax is correct. This is checked by the XML parser for all XML documents and is not the XML schema's responsibility. XML documents that are well formed and abides to all other constraints, like those introduced by XML Schemas, are considered *valid*. XML schemas defines constraints and rules that go beyond the XML syntax. They can define exactly what elements and attributes that must be used by a document, how many times elements should occur, ordering of elements, etc. They can even define the data type of individual elements by using a standardized set of data types that is part of the W3C XML Schema specification[13].

What makes XML Schemas useful is that they can constitute a contract between one or more parties for how data should be described in XML. As long as the XML schema does not change the parties that exchange data can be certain that the data received will be in the correct format and use the right data types. This reduces the amount of extra error handling significantly since validity can be automatically checked by the XML parser. Furthermore, XML Schemas can be designed to easily be extendible, flexible and modular with respect to changes. This can be accomplished[14] by using encapsulation, inheritance, and polymorphism techniques known from object-oriented programming languages when you first design your schema.

When creating XML Schemas it is useful to know certain design strategies to create more decoupled and reusable schemas. There are currently three design patterns that define different levels of granularity concerning the components a schema is made up of:

- Russian Doll
- Salami Slice
- Venetian Blind

Russian Doll is the least extensible design where you do not reuse any types, but wrap all necessary components within each other and just redefine any duplicate types. Salami Slice is more extensible because you define elements separately and reference these when more complex elements are composed of other elements. This design allows for more granularity than Russian Doll because elements can be reused. Venetian Blind is the most extensible design because it allows reuse of both elements and types. All types and elements are defined in the global scope

[12] http://www.informit.com/articles/article.asp?p=102160&redir=1&rl=1
[13] http://www.w3.org/XML/Schema
[14] http://www-128.ibm.com/developerworks/library/x-flexschema/

so that these can be used for defining new elements or types. An example of the Venetian Blind design can be seen in listing 4.1.

```
<xs:simpleType name="Title">
    <xs:restriction base="xs:string"/>
</xs:simpleType>

<xs:simpleType name="Name">
    <xs:restriction base="xs:string">
        <xs:minLength value="1"/>
    </xs:restriction>
</xs:simpleType>

<xs:complexType name="Book">
    <xs:sequence>
        <xs:element name="Title" type="Title"/>
        <xs:element name="Author" type="Name"/>
    </xs:sequence>
</xs:complexType>
```

Listing 4.1: Venetian Blind Example

It may seem like the Venetian Blind design pattern is best because it allows the most extensibility, but this is not always the case. Usually a developer will try to find a good mix between these patterns. If you always reuse all your types and elements it might end up being cumbersome to change them in the future. This can happen if you reuse components that are identical when you create the schema, but that may become different when later changes are incorporated.

4.5.2 XML Bindings

XML Binding tools are tools that can map XML to and from objects in a specific programming language. By using such a tool developers do not have to work with XML directly, instead they can just deal with the objects produced by the mapping tool in their favorite programming language. The tools usually create classes for a specific programming language by using an XML schema. The created classes can then hold all the information from XML documents that are constrained by the schema. Most XML Binding tools rely on XML schemas to do the mapping automatically, but some tools allow you to map the XML to your own classes.

However, one of the problems associated with mapping an XML schema to classes in a programming language is that the classes cannot capture all the constraints specified by a schema. This limitation forces developers to add manual checks for the constraints and relationships that get lost in the mapping process. Furthermore, optional elements and attributes often specified in XML schemas are

difficult to deal with in an object-oriented environment. The different XML Binding tools all have different ways of solving these problems.

For Java there exists a number of different XML Binding tools that each have their own strengths and weaknesses. Some of the popular ones are JAXB[15], Castor[16] and XMLBeans[17]. Microsoft provides only one tool for .NET called the Schema Definition Tool (XSD.exe)[18] that maps an XML schema to C# classes.

JAXB

JAXB is the reference implementation of Sun's Java API for XML Binding specification. It provides a simple way to generate Java classes from an XML schema by using a command-line tool. One of JAXB's strengths is that it allows users to customize various aspects of the generated data binding. These customizations can either be applied though annotations in the schema document or by the use of an external binding declaration document that is passed to the command-line tool. The customizations include[19]:

- Options for controlling the names of generated classes and properties
- A way to specify an existing implementation class to be used by the binding
- Options that allow (limited) control over the validation handling and the serializers/deserializers used for marshalling and unmarshalling

Microsoft Schema Definition Tool

The Microsoft Schema Definition Tool also uses a command-line tool (XSD.exe) for generating C# classes based on an XML schema. While it does not provide all the customization capabilities as JAXB it is considerably simpler to use the customizations that is provided. It does not rely on any annotations in the schema or external binding declaration documents, instead it has all options easily available in the command-line tool.

However, the Schema Definition Tool has some more serious flaws. For one thing it bases the class names directly on the element names stated in XML schemas. Since it is common to start XML element names with lowercase letters this will cause the Schema Definition Tool to generate classes that also start with lowercase letters. There is no workaround for this except changing your XML element

[15] https://jaxb.dev.java.net/
[16] http://www.castor.org/
[17] http://xmlbeans.apache.org/
[18] http://msdn2.microsoft.com/en-us/library/x6c1kb0s(vs.71).aspx
[19] http://www-128.ibm.com/developerworks/library/x-databdopt/

names to start with uppercase letters. While this is issue is annoying, a bigger problem is that the Schema Definition Tool does not support all data types specified in the XML Schema specification. For instance, JAXB supports the `<xs:duration>` data type for XML elements. This complex data type specifies a duration in time using a format defined by the XML Schema specification. JAXB maps this data type to a Java class specifically designed for time periods, but the Schema Definition Tool just maps it to a C# String. A consequence of this is that you have to check what data types are supported before you use the tool and find workarounds for those data types that are not supported.

HyperJAXB

HyperJAXB[20] is not a XML binding tool, but a tool that can be used in conjunction with JAXB to automatically create hibernate mapping files (.hbm.xml) for JAXB objects. This means that JAXB objects can, with very little effort, be stored persistently in a database through Hibernate. Considering that the most difficult part of Hibernate is to get the mapping files right, a tool that generates them automatically (even just for JAXB classes) would be very practical.

Furthermore, HyperJAXB is very useful in scenarios where you:

- Work with XML documents that are tied to an XML schema
- Use JAXB to work with those documents
- Have an XML schema that is likely to change

In situations where the XML schema is likely to change you want a system that can easily adapt to that change. If you use HyperJAXB together with JAXB you can run a single script to generate both updated JAXB class files and hibernate mapping files to enable the JAXB objects to be stored using Hibernate.

Although HyperJAXB have some very nice features it also has some big limitations. The biggest is that the only version that works properly is written for JAXB 1.0 and Hibernate 2.0. JAXB is currently in version 2.0 and Hibernate in version 3.0 and it is not worth downgrading just to use this functionality. A new version is being developed to support the latest Hibernate and JAXB versions, but it is still in its infancy and not ready for use.

[20] https://hyperjaxb.dev.java.net/

4.6 Summary

This chapter has described a number of different tools and technologies that have played crucial roles in this thesis. Most importantly it has given an introduction to the Argos middleware platform and outlined what features it provides. The following chapters will refer to this chapter when describing how the technologies are actually used.

Chapter 5

Design

This chapter will present the design for the various system components based on the requirements that were determined in the requirements chapter.

5.1 Overview

The requirements from chapter 3 divided the system into three main components; The Sensor Configuration Tool, The Argos Sensor Service and the Mobile Sensor Framework. This chapter will give a high level overview on how these components relate to each other and present a detailed architecture for each of them.

5.2 High level architecture

This thesis will develop functionality for the Argos middleware platform that was described in section 4.3 of the technology chapter. The Sensor Service created in this thesis is developed as a Argos *system service*. It will provide functionality that allows Argos applications to more easily access sensors and sensor data. Figure 5.1 attempts to illustrate how the Argos middleware platform is designed and how the Argos Sensor Service will fit in as a new system service. The illustration also shows other system services that the Argos Sensor Service will use and depend on. Outside of Argos the Sensor Configuration Tool and Mobile Sensor Framework are displayed as large boxes that interact with the Sensor Service. The boxes with red borders contain functionality that will be developed in this thesis.

The Mobile Sensor Framework should provide reusable functionality for interacting with sensors and provide ways to distribute sensor data to the Argos Sensor

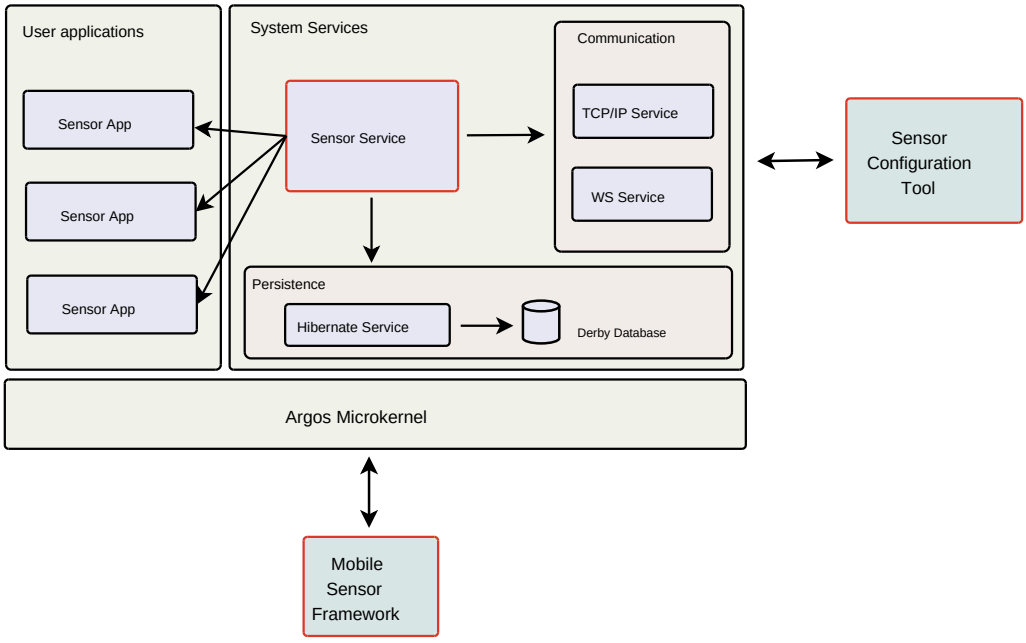

Figure 5.1: High level architecture of the system

Service. To accomplish this it will need configuration information on how to interact with specific sensors and how to communicate with Argos. Much of this information will change depending on the sensor that is used and must therefore be provided by people with the proper technical knowledge. The Sensor Configuration Tool will simplify the collection of this information by providing an easy to use GUI to fill out the necessary data. The tool should be able to produce a configuration in the correct format and allow a user to deploy the configuration to a device running the Mobile Sensor Framework. The Argos Sensor Framework should take care of the details concerning the deployment of configurations and provide ways to receive sensor data from devices that interact with sensors.

By separating the GUI from the Argos System Service functionality and using the Hibernate system service provided by Argos to store data, the sensor system created for this thesis employs a *three-tier architecture* as illustrated by figure 5.2. The main advantages of using such an architecture includes increased flexibility, maintainability and reuseability as described by [20].

5.3 Identifying common sensor functionality

One of the oldest and most productive development strategies have been to identify and extract commonly used functionality in order to create reusable and

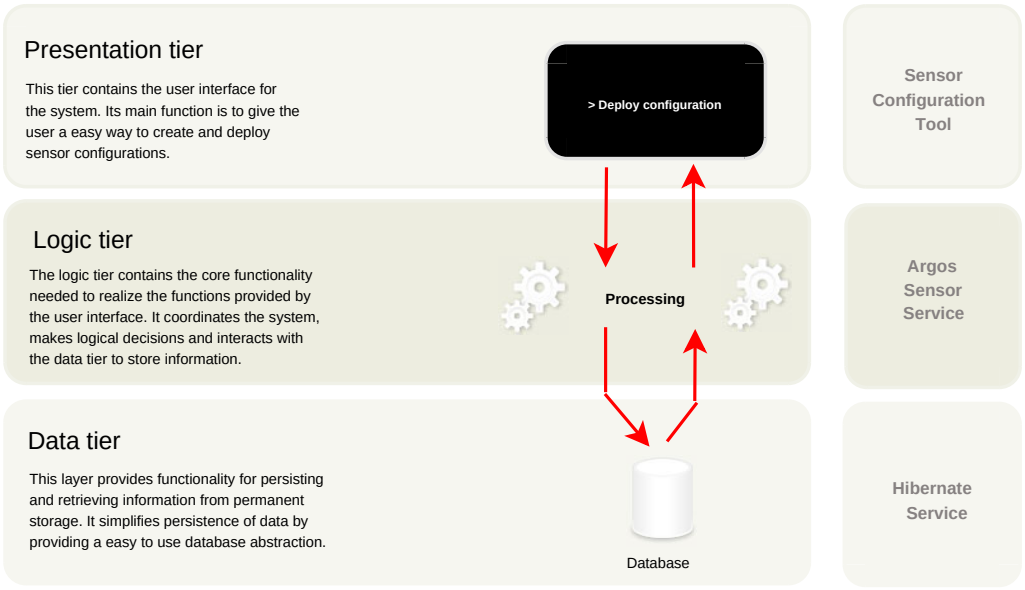

Figure 5.2: Tier architecture of the system

reliable software libraries. In this thesis this approach has been applied to develop functionality that is common to applications that interact with sensors and use sensor data. In a scenario where two applications are developed from scratch to collect sensor data from two different types of sensors, some of the functionality would most likely be fairly similar. They might, for instance, use the same technologies for communication, they might both convert the native data format of the sensor to their own format or they might both want to send the data to a remote location for processing. Hence it would be a good idea to extract this functionality into some sort of library to enable future sensor applications to be developed faster and more reliably.

While a library containing common functionality for accessing sensors would allow sensor applications to reuse a lot of functionality it is possible to extend on this idea to accomplish more. What if the library could be run as a stand alone application and dynamically accept ”descriptions” that enabled it to talk to and extract information from any type of sensor? It would not be necessary to develop a whole new application to access a new type of sensor. As long as a device or computer ran this ”library” or framework as a application it could be reconfigured dynamically to access and distribute data from any sensor to remote locations. This is the general idea behind the Mobile Sensor Framework, which will be described later in this chapter.

5.3.1 Describing sensor functionality

The problem with creating such a stand alone application/library is describing the functionality that is specific to each sensor type. Typically this information is related to how to communicate with a sensor and how to convert its native data format into a more generic format as was discussed in 2.2 and in 2.3.

In an ideal world this information should be described in a interoperable format so that it could be reused across different programming languages and platforms. The library itself would still have to be developed for different platforms, but the information regarding each sensor type would only have to be described once. However, as discussed in 2.2, it is not trivial to capture such information in an interoperable format. To achieve complete interoperability across platforms and programming languages you are pretty much forced to describe information in some kind of markup language[1]. Using a markup language means that a standard format for describing this type of information has to be defined. Unfortunately due to the diverseness of sensors it would be extremely complicated to define a format that could capture this information for any sensor.

The alternative, which is adopted by this thesis, is to specify this information as modules in a specific programming language. This approach has the obvious drawback of having to redevelop the modules if other programming platforms are to be used, but it will ensure, without a doubt, that it is possible to describe the information for any type of sensor. Furthermore, it might enable future researchers to analyze programming modules for several sensors to attempt to discover if this information indeed can be mapped to a markup language instead.

5.3.2 Sensor Protocol Plugins

In the requirements chapter use-case T-8 stated that a sensor profile must contain one or more sensor interaction descriptions. A sensor interaction description was defined to *provide the information necessary to extract and interpret the measurements the sensor provides.* The purpose of these descriptions is to capture the information specific to different types sensors, as discussed in the previous two sections.

Since it was decided in section 5.3.1 that this information should be specified as modules in a programming language they will be referred to as *Sensor Protocol Plugins* for the rest of this thesis.

[1] http://www.computerworld.com/action/article.do?command=viewArticleBasic&articleId=103711&pageNumber=1

5.4 The Sensor Configuration Tool

5.4.1 Overview

The Sensor Configuration Tool will, in combination with the Argos Sensor Service, provide functionality for creating sensor profiles and sensor configurations. As seen in figure 5.1 this tool will run independently of Argos and will only provide the user interface for interacting with the Argos Sensor Service. These system components could easily be merged into a single application/service running within Argos, but there exists several good reasons for why developing them as separate applications is a good idea. The number one reason is that it gives a good separation of functionality. The user interface should only be concerned with collecting sensor profile and configuration information from a user while persistence and deployment is taken care of by the Argos Sensor Service. This design will also allow the Sensor Configuration Tool to access the Argos Sensor Service remotely since it would not be any more difficult than interacting locally. Interacting remotely will be much slower than locally, but this will not be an issue since only small amounts of information needs to be exchanged.

Figure 5.3 shows a very high level overview of how the Sensor Configuration Tool is designed. When a user starts the tool he must connect it to a Argos that is running the Argos Sensor Service. The Argos may be running locally or remotely, it should not matter as far as the Sensor Configuration Tool is concerned.

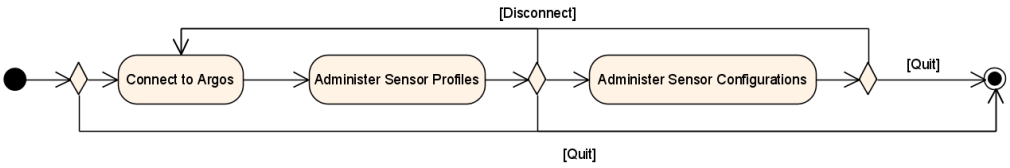

Figure 5.3: Overview of the Sensor Configuration Tool

5.4.2 Managing Sensor Profiles

The activity diagram in figure 5.4 illustrates in more detail what should happen after the Sensor Configuration Tool is connected to the Argos Sensor Service. When connected, the Sensor Configuration Tool should fetch any existing sensor profiles from the Argos Sensor Service and display these to the user. Afterwards the user should be allowed to add new profiles or view, edit or delete selected profiles.

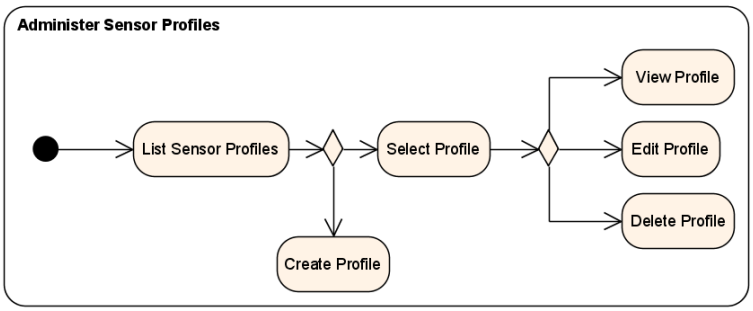

Figure 5.4: Managing sensor profiles with the Sensor Configuration Tool

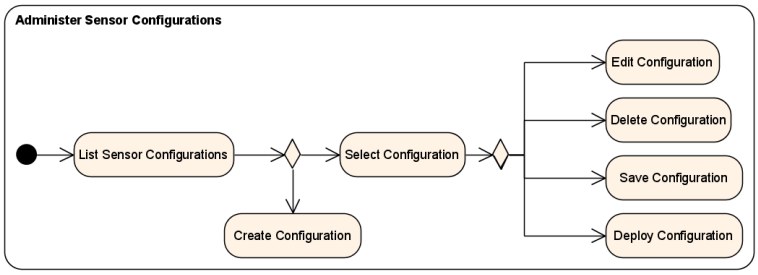

Figure 5.5: Managing sensor configurations with the Sensor Configuration Tool

5.4.3 Managing Sensor Configurations

After a user has selected a sensor profile it will be possible to create and select sensor configurations. Figure 5.5 illustrates how sensor configurations are to be administered. When a sensor profile is selected all sensor configurations that belong to this profile should be displayed to the user. The user will then be given the option to create a new sensor configuration based on a selected sensor profile or he can select an already existing configuration to edit, save, deploy or delete. Upon deployment sensor configurations will be passed on to the Argos Sensor Service. It will store the configuration and deliver it to the device specified by the Sensor Configuration Tool. The Sensor Configuration Tool will also make it possible to save any (deployed or not) configuration to a local file. This will enable the user to deploy the configuration manually if he or she wishes.

5.4.4 Deploying Sensor Configurations

Deploying a sensor configuration by using the Sensor Configuration Tool will make it collect all necessary information from the user interface and generate a configuration file in an interoperable format. The Sensor Configuration Tool should also verify that the format of the produced configuration is correct. Finally the configuration file will be delivered to the Argos Sensor Service, which will take

care of deploying the configuration to the device that is connected to the sensor in question. These activities are illustrated in figure 5.6.

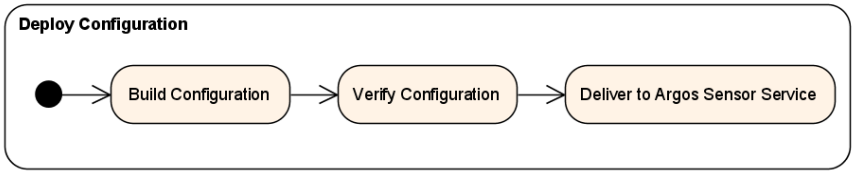

Figure 5.6: Deploying sensor configurations with the Sensor Configuration Tool

5.4.5 Putting the pieces together

The previous sections presented the design for the Sensor Configuration Tool by dividing the user interface into smaller, more manageable components. Figure 5.7, in this section, shows how these pieces fit together in a more complete diagram.

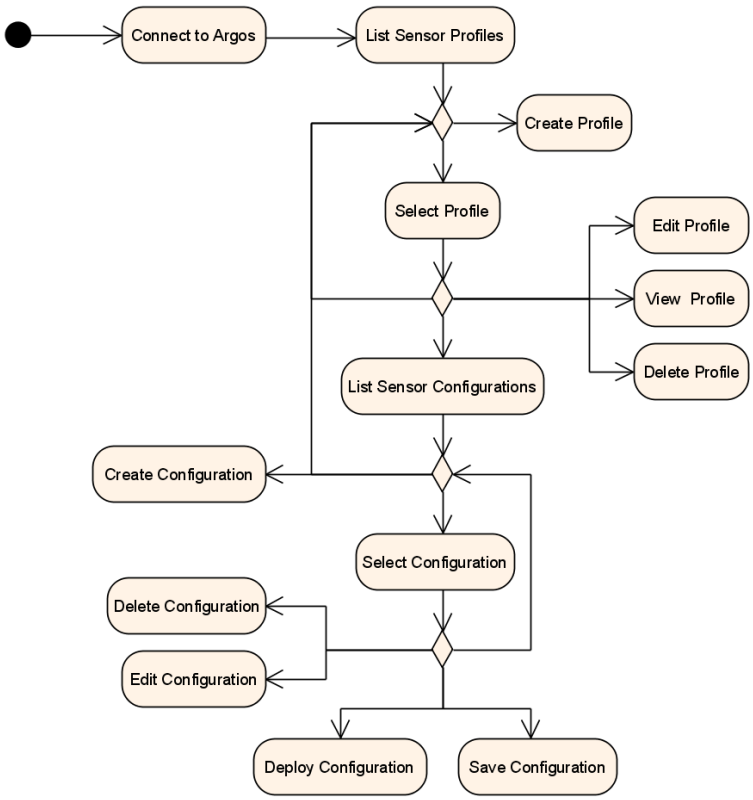

Figure 5.7: A complete overview of the Sensor Configuration Tool

5.5 The Argos Sensor System Service

5.5.1 Sensor Profiles

The Argos Sensor Service should act as a repository for sensor profiles. This means that Sensor Profiles created by the Sensor Configuration Tool should be permanently stored (until deleted) by the Argos Sensor Service. When the Sensor Configuration Tool first connects to the Sensor Service it will fetch all the profiles that are stored and display these to the user. The user can then choose to create a sensor configuration based on one of the fetched profiles.

Figure 5.8 shows a simple activity diagram showing a sensor profile being received and stored by the Argos Sensor Service. When receiving a sensor profile it also receives one or more sensor protocol plugins belonging to the profile. As explained previously in section 5.3.2 these plugins represent modules in specific programming languages that "knows" how to communicate and extract information from a sensor.

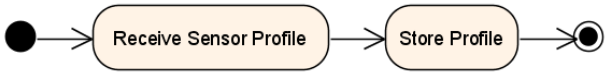

Figure 5.8: Storing sensor profiles

5.5.2 Sensor Configurations

One of the primary functions of the Argos Sensor Service is to deploy sensor configurations. Figure 5.9 illustrates the deployment of a sensor configuration by expanding on the chain of events that were shown in section 5.4.4. As shown in the illustration the Argos Sensor Service first receives a configuration from the Sensor Configuration Tool and stores it locally. Then it notifies Argos user applications and the device connected to the sensor that a new or modified configuration is available. When the configuration is stored the Argos Sensor Service should also generate a unique ID to assign to the configuration. This ID should be propagated to the user applications and the device when they are notified.

The device receiving the notification should be running the Mobile Sensor Framework and should afterwards contact the Argos Sensor Service to get the actual configuration. More details about this will be given in the design of the Mobile Sensor Framework.

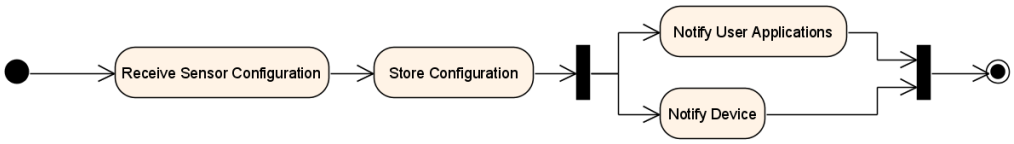

Figure 5.9: Deploying Sensor Configurations

5.5.3 Receiving Sensor Data

The Argos Sensor Service is intended to receive sensor data collected by devices that are directly connected to sensors and distribute this information to Argos user applications. This functionality is illustrated by figure 5.10. Argos user applications should register with the Argos Sensor Service in order to receive sensor data from a particular sensor.

Figure 5.10: Receiving and distributing sensor data

5.6 The Mobile Sensor Framework

The Mobile Sensor Framework is a library/application that should provide common functionality to interact with sensors. It is an attempt to realize the ideas that were discussed in section 5.3. Similar to the functionality talked about in that section, the Mobile Sensor Framework should act as both a library and as a stand alone application.

The main idea is that this framework can be given sensor protocol plugins that allows it to interact with- and collect data from any type of sensor. Furthermore, the framework should provide ways to distribute the collected sensor data to remote receivers such as a remotely running Argos Sensor Service. In order to connect to- and extract data from a specific sensor the framework needs more information than just a sensor protocol plugin. It needs all configuration details on how to connect to the sensor and information related to how and where the sensor data should be distributed. All this information should be provided by sensor configurations created by the Sensor Configuration Tool.

Figure 5.11 illustrates a device running the Mobile Sensor Framework getting a sensor configuration and sensor protocol plugin from the Argos Sensor Service.

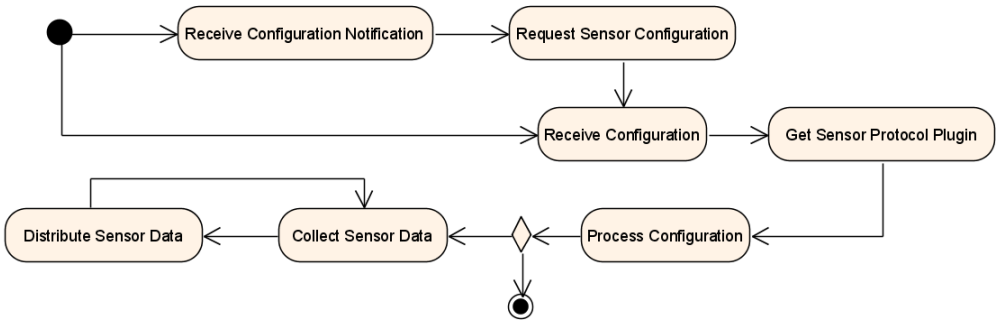

Figure 5.11: Getting sensor configurations and sensor protocol plugins

Since this is a framework for mobile devices there is no way to send the configuration or the protocol plugin directly to the device. This problem was discussed previously in section 2.5 about handheld devices. Instead an SMS notification must be sent to the device that lets it know how it can retrieve the sensor configuration. The configuration should then contain information on how the device can retrieve the sensor protocol plugin.

However, as illustrated by the figure, it should also be possible for the Mobile Sensor Framework to receive a sensor configuration directly. As just described this is not possible remotely, which means that a configuration must be delivered to the Mobile Sensor Framework locally. This can happen when another mobile application uses the Mobile Sensor Framework as a library. The mobile application may want to give the Mobile Sensor Framework a sensor configuration it already is in the possession of.

5.7 Summary

This chapter has focused on the design and operation of the Sensor Configuration Tool, Argos Sensor Service and the Mobile Sensor Framework. An attentive reader might have noticed that no details have been provided yet about what information sensor configurations and sensor profiles actually contain. This information will be described in depth in the next chapter.

Chapter 6

Implementation

This chapter will describe the implementation details of the completed system. Before each system component is outlined the implementation environment and technologies used will be presented.

6.1 Environment and programming language

The Argos middleware platform along with all its user- and system services is implemented in Java. Since the Argos Sensor Service is a system service it obviously needs to use the same language. A different language could potentially be picked for implementing the Sensor Configuration Tool. However, as we will also see later in this chapter, communication is likely to be far easier between two applications that are implemented in the same programming language. For this reason the Sensor Configuration Tool is also developed in Java.

The Mobile Sensor Framework is intended to be run on some sort of handheld, mobile device. The only requirement for the device is that it must support Bluetooth and IR and that it can access the Internet. Bluetooth and IR are required for interacting with the demonstration sensors that are available.

It is possible to implement the Mobile Sensor Framework using any of the many different mobile platforms available, but J2ME[1], the limited edition of Java for such devices, seems like the obvious choice. The advantage of J2ME is that applications can run on the wide range of devices that supports Java and in our case it probably makes communication easier since the rest of our architecture uses Java. Despite these advantages J2ME was not picked for implementation.

[1] http://java.sun.com/javame/index.jsp

Instead Microsoft's Windows Mobile 5.0 (WM5) platform was chosen using the .NET Compact Framework programming language.

The primary reason for using WM5 was that NST uses this platform and that cell phones with WM5, as a result, was readily available. Also, since requirement NFR-106 states that the Mobile Sensor Framework and Argos Sensor Service should be loosely coupled, implementing these components using different platforms will demonstrate that our system indeed complies with this requirement.

6.2 System Overview

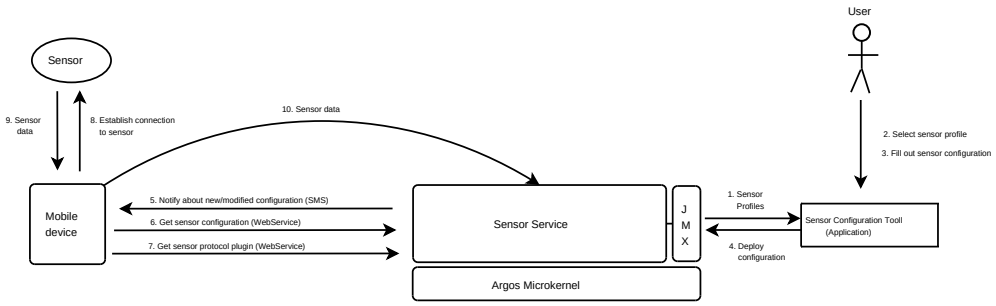

Figure 6.1: Overview of information flow in the system

Figure 6.1 illustrates of how the different system components communicate with each other. The Sensor Configuration Tool interacts with the Argos Sensor Service using JMX. As described in section 4.4.5 JMX enables services to expose methods through JMX to remote clients. JMX uses RMI underneath, but hides most of the details of how the remote procedure calls are performed.

The Sensor Service running in Argos uses SMS messages to notify mobile devices of new or changed sensor configurations. Mobile devices that receive a SMS notification use a Web Service exposed by the Argos Sensor Service to get the sensor configuration associated with the notification. The retrieved configuration will specify a sensor protocol plugin that the mobile device needs to communicate with the sensor. If the mobile device has not previously retrieved the sensor protocol plugin, it will get it by using another Web Service exposed by the Argos Sensor Service.
When the mobile device has both the sensor configuration and the sensor protocol plugin it will decide based on the configuration what to do next. The configuration will specify if the sensor data should be collected right away and how often it should be collected. If the sensor should be contacted immediately the Mobile Sensor Framework running on the device will spawn a new thread that will be

responsible for collecting the sensor data at the intervals specified. The thread will use the sensor protocol plugin to do the actual sensor interaction and data extraction. Every time new sensor data has been collected, the data is converted to a standard sensor format specified by the Argos Sensor Service. The details of this format will be given later in this chapter. When data from the sensor has been extracted and converted to the appropriate format the Mobile Sensor Framework will deliver it to the Argos Sensor Service.

6.3 Distributing sensor data

Section 2.5.2 in chapter 2 discussed how sensor data could be distributed from handheld devices to remote receivers. The benefits and drawbacks of push- and pull-based approaches were compared, but an approach for this system was not chosen. During the initial phases of development supporting both approaches was considered, but it was realized early on that this would involve too much work. In the end the push-based approach was adopted. The reason was mainly because of the limitations associated with polling handheld devices, but it would also require less work to implement than a pull-based approach.

In order for a handheld device to push data to a receiver, the receiver must provide a service that can accept the data. The service could typically be either a Web Service or a tcp/ip server. A Web Service would be the easiest approach since a Web Service is fairly easy to set up both at the client side and server side and receiving data would not require any additional processing. However, the ease of use has a performance penalty as described by [22]. The reason for this penalty is mainly that Web Services use SOAP[2], which is a XML based, interoperable protocol for exchanging messages over a computer network. Since SOAP wraps messages in its own XML format it will add both space overhead and parsing overhead to a Web Service call. Like discussed in 2.3.1 the size overhead is likely to be on average around 400%. For handheld devices Web Services also require significant computational resources which will contribute to reducing battery life and might affect the responsiveness of other applications.

The alternative is to use raw tcp/ip sockets. Web Services also uses tcp/ip underneath, but has protocol layers like HTTP and SOAP on top of it. Using raw sockets gives you great flexibility and performance at the cost of having to create your own protocol. However, it is relatively simple to create a tcp/ip protocol to just send packets consisting of XML formatted sensor data, although not as simple as just calling a Web Service. If you do not need the extra performance you would use a Web Service because it would give you the added benefit

[2]http://www.w3.org/TR/soap12-part0/

of interoperability, but if the device or platform makes using Web Services difficult/impossible then tcp/ip sockets would be the way to go. Since Argos provides an easy to use tcp/ip service it was decided to support both approaches.

6.4 The Sensor Data Format

The design chapter in section 5.3.1 briefly mentioned that the native data format of a sensor should, at some point, be converted into a more generic format. When measurements are extracted from a sensor the data is usually contained in a compact proprietary format defined by whoever manufactured the sensor. Section 2.3 discussed different technologies that could be used for creating a format to generically describe sensor data. It also mentions that due to the diversity of sensors it might not even be possible to define such a format.

Nevertheless, based on this discussion a decision was made to try to define a XML format that could describe sensor data as generically as possible. This XML format is defined by the sensor packet XML schema illustrated in figure A.1 in appendix A. When data is collected from sensors by the Mobile Sensor Framework, it will convert the data into this format before it is sent to the Argos Sensor Service and/or other receivers.

The format defined is based on the measurements produced by a very limited number of sensors. A larger number of sensors should have been investigated, but it was just not possible due to the limited time available. The sensors that were used include:

- The EMTAC CRUXII Wireless GPS Sensor[3]
- The Davis Weather Station
- A custom step sensor created by NST

6.4.1 The EMTAC CRUX II Wireless GPS Sensor

The EMTAC GPS Sensor is a wireless GPS receiver that broadcasts position measurements using bluetooth. Although the most useful information it provides is latitude and longitude readings, it also produces a number of other measurements as defined by the NMEA standard. The NMEA 0183 standard[4] is defined by the National Marine Electronics Association (NMEA)[5] and is a proprietary data specification for communication between marine electronic devices such as

[3] http://www.emtac.com/support/cruxsupport.html
[4] http://www.nmea.org/pub/0183/
[5] www.nmea.org/

depth finders, navigation instruments, and GPS receivers. In addition to hardware protocol specifications the NMEA standard defines a number of different *sentences* that each has its own unique interpretation. The sentences describe all kinds of information related to marine environments and some of them describe GPS related data.

Since the NMEA 0183 standard is proprietary it is not available for free. It can be purchased from the NMEA association, but this costs several hundred dollars. Fortunately there exists many web sites[6,7] that provide unofficial explanations on which NMEA sentences exist and how to interpret them. These web sites contained enough information to figure out which sentences the EMTAC GPS Sensor supports and how they should be deciphered. The NMEA sentences broadcasted by the EMTAC GPS Sensor are:

- GGA - Global Positioning System Fix Data
- GSA - GPS Dilution Of Precision (DOP) and Active Satellites
- GSV - GPS Satellites in View, Elevation and Azimuth
- RMC - Recommended Minimum Specific GPS/TRANSIT Data

Each of these sentences contain multiple pieces of information. For instance, the GGA sentence contains fifteen different fields, some of the which are:

- Sentence Identifier
- Time
- Latitude
- Longitude
- Fix Quality (0=Invalid, 1=GPS fix, 2=DGPS fix)
- Number of satellites in use
- Altitude
- Time since last DGPS update

The generic XML sensor data format must be able to describe all information contained in the different NMEA sentences. It must also be able to describe multiple sentences in a single document. The EMTAC GPS Sensor has the most complex and diverse measurements of the three sensors even though it is a relatively simple sensor. For this reason the generic sensor format was first created

[6] http://www.gpsinformation.org/dale/nmea.htm
[7] http://www.kh-gps.de/nmea-faq.htm

to describe the measurements of the EMTAC GPS Sensor and later retrofitted to work with the other sensors. Figure 6.1 shows a excerpt from the finished XML sensor data format describing the GGA NMEA sentence information. The complete example can be found in appendixA.

```
<?xml version="1.0"?>
<sensorPacket id="21323423" date="2007-02-15" time="12:04:10">

   <instrument name="GGA">
      <measurement name="Time" unit="time" value="170834"/>

      <measurementGroup name="Latitude">
         <measurement name="Degrees" unit="degrees" value="41"/>
         <measurement name="Minutes" unit="minutes" value="24"/>
         <measurement name="Seconds" unit="seconds" value="54"/>
         <measurement name="Direction" unit="direction" value="north"/>
      </measurementGroup>

      <measurementGroup name="Longitude">
         <measurement name="Degrees" unit="degrees" value="81"/>
         <measurement name="Minutes" unit="minutes" value="51"/>
         <measurement name="Seconds" unit="seconds" value="41"/>
         <measurement name="Direction" unit="direction" value="west"/>
      </measurementGroup>
   </instrument>

</sensorPacket>
```

Listing 6.1: EMTAC GPS XML Data format Example

6.4.2 The Davis Weather Station

The Davis Weather Station was previously used as an example in section 2.4.1 about SensorML. Figure 2.2 in that section also gave a high level overview of the measurements the station provides. The reason this sensor system was used is that the SensorML web site provides a full SensorML profile describing the station. This profile contains all information concerning the Davis Weather Station including details regarding its measurements. Listing B.2in appendix B shows the measurement information given by the profile.

Our completed XML sensor data format can describe the measurements provided by the station according to how they are specified by the SensorML profile. Figure 6.2 shows a excerpt from an example description using our XML format. The complete example can be found in appendix B.

```
<?xml version="1.0"?>
<sensorPacket id="21323423" date="2007-02-15" time="12:04:10">

   <instrument name="weatherMeasurements">
      <measurement name="Time" unit="iso8601" value="20071231T235959"/>
      <measurement name="Temperature" unit="celcius" value="59"/>
      <measurement name="BarometricPressure" unit="bar" value="120"/>
      <measurement name="WindSpeed" unit="meterPerSecond" value="25"/>
      <measurement name="WindDirection" unit="degree" value="180"/>
      <measurement name="RainFall" unit="meter" value="10"/>
   </instrument>

</sensorPacket>
```

Listing 6.2: Weather Station XML Data Format Example

The example shows that the data format for the Davis Weather Station is easier to represent than that from the EMTAC GPS Sensor. However, based on the definition of the data format given by the SensorML profile it is also apparent that there exists many ways to map the format given by the profile to our format. Ideally it should be just one way to do the mapping, but this is a limitation of creating such a generic format. For this prototype the main concern is that the measurements can be represented without losing information. Note that even though the SensorML profile also provides phenomenon definitions this information is not necessary for describing the sensor data.

6.4.3 NST Step Sensor

The NST Step Sensor was mainly included because NST might want to further develop the prototype system created in this thesis. It is a very simple sensor that provides a single Step measurement. Listing 6.3 shows how this data can be described in our generic sensor data format.

```
<?xml version="1.0"?>
<sensorPacket id="21323423" date="2007-02-15" time="12:04:10">

   <instrument name="StepSensor">
      <measurement name="Steps" unit="step" value="2560"/>
   </instrument>

</sensorPacket>
```

Listing 6.3: NST Step Sensor Data Format Example

6.5 System Components

This section will outline the implementation details of the various system components. Due to the size and complexity of the system only the most important

aspects of each component will be described.

6.5.1 The Sensor Configuration Tool

The Sensor Configuration Tool allows users to create sensor profiles and sensor configurations by interacting with the Argos Sensor Service. The tool is developed as an independent Java Swing application, but is only intended to be a user interface for the Argos Sensor Service. Its basic operation is to collect data from the GUI, map this information to the sensor configuration data format or the sensor profile data format and send it to the Argos Sensor Service for further processing.

Representing sensor profiles and configurations

A Sensor profile contains information specific to a sensor type while a sensor configuration contain information on how to communicate with a real sensor. Profiles and configurations are created by the Sensor Configuration Tool, stored and deployed by the Argos Sensor Service and used by the Mobile Sensor Framework. In order for this information to be shared across different programming languages and platforms XML was adopted to provide a common, interoperable formats.

Two separate XML schemas were created for defining the details on how the sensor profile and sensor configuration information should be represented in XML. Chapter 4 discussed the benefits of such schemas and also presented how XML schemas should be designed. A combination of the Salami Slice and Venetian Blind design patterns were used to separate most of the information into components that can easily be altered or completely replaced if the schemas need to be changed in the future. Chapter 4 also presented XML Binding tools to automatically map schemas to objects in different programming languages. Mapping schemas to objects not only makes it easier to work with the XML, but it also makes it easier to change the formats later since updated objects can be recreated instantly. To make it as easy as possible to change the different XML formats used by the system automatic scripts were created to generate all necessary classes. The scripts use JAXB and the Microsoft Schema Definition Tool described in chapter 4 to generate both Java and C# classes with the click of a button. As long as changes to the schemas are not major the generated classes can just be inserted directly into each application and relatively minor corrections need to be made.

A visual representation of the sensor configuration schema and the sensor profile schema can be found in appendix A.

Creating a sensor profile

Sensor profiles are created using the Sensor Configuration Tool GUI and stored by the Argos Sensor Service. A sensor profile contains the following items:

- A sensor type identification
- One or more instruments
- Zero or more of sensor options
- One or more sensor protocol plugins

The sensor type identification is used to specify which type of sensor the profile is for and will uniquely identify a sensor profile in our system. The scheme adopted is partly based on how SensorML identifies sensors and consists of the following fields:

- The fully qualified name of the sensor
- The sensor type
- The sensor model
- The sensor manufacturer

The name, model and manufacturer fields are taken directly from the SensorML specification. The type was added to make it possible for Argos applications to filter/select which sensor they want to collect data from based on its type. Note that functionality for doing filtering was never implemented, but remains possible to be added in the future.

A sensor profile has one or more instruments that must be added through the GUI. This information is intended to describe how sensor data from the sensor is mapped to the generic sensor packet format described in 6.4. Developers of sensor protocol plugins will use this information to do the proper mapping after collecting sensor data, which will enable developers of Argos applications to just look at the sensor profile to see how to interpret the sensor data.

A sensor profile can have zero or more sensor options that are related to the type of sensor the profile describes. Due to time constraints a sensor option in this prototype consist only of a name, but in the future it should also specify the type and unit of data. The sensor options in a profile are used to determine what information a sensor configuration shall contain. That is, a configuration based on a specific sensor profile will allow users to assign values to the sensor options

defined by that profile.

Last, but not least, a sensor profile must contain one or more sensor protocol plugins. The purpose of a sensor protocol plugin was described in section 5.3.2. It is basically a module in a programming language that provides the functionality necessary for the Mobile Sensor Framework to connect to and collect measurements from a specific type of sensor. A protocol plugin added to a sensor profile should obviously contain the functionality necessary to access the type of sensor the profile describes. The Mobile Sensor Framework developed in this thesis uses the .NET Compact Framework, consequently protocol plugins must be developed for this platform. A protocol plugin written in C#for the .NET Compact Framework consists of a .dll library file that must be added to a sensor profile through the GUI. Sensor profiles allows several protocol plugins in different programming languages to be added in order to work with Mobile Sensor Frameworks developed for other platforms.

Creating a sensor configuration

Sensor configurations contain all configuration information the Mobile Sensor Framework needs to communicate and collect sensor data from specific sensors. When creating a configuration using the Sensor Configuration Tool some additional information is also specified that is only used by the Argos Sensor Service.

A sensor configuration is based on a specific sensor profile that must be selected in the user interface before the configuration can be created. Sensor configurations produced by the Sensor Configuration Tool contains the following items:

- A name (Just for naming the configuration)
- A name for the sensor that is unique to the Mobile Sensor Framework
- A sensor profile
- A start/stop status
- An remote binding
- Sensor options
- A frequency specifying how often sensor data should be retrieved

The name given to the sensor is used by the Mobile Sensor Framework to distinguish between multiple sensors that might be connected. It is also the sensor name used in the sensor data packets that are sent from the Mobile Sensor Framework to the Argos Sensor Service. The configuration will be linked to a specific

sensor profile by accommodating the sensor profile identification specified in the previous section. The Mobile Sensor Framework uses this information to retrieve the sensor profile and sensor protocol plugin from the Argos Sensor Service after it has received the configuration.

The start/stop status specifies if sensor data should be collected or not. If a sensor configuration was issued to start collecting sensor data a second configuration can be issued to stop the collection.

The configuration must contain a remote binding that specifies where and how sensor data should be delivered. In configurations created by the Sensor Configuration Tool the destination is always the computer the tool is connected to, but with relatively minor changes the data could be sent to a completely different location. The Sensor Configuration Tool allows users to choose between delivering the data using a Web Service or a tcp/ip server. A fixed Web Service address is specified for the prototype, which means that servers intending to receive sensor data using a Web Service must follow these conventions. The only options that must be specified in the configuration is the ip/host address of the receiving server and the port if a tcp/ip server is used. For simplicity the Sensor Configuration Tool does not allow the user to specify the ip/host address, instead it just uses the address entered by the user to connect the tool to the Argos Sensor Service. This address must be globally reachable since it is the device running the Mobile Sensor Framework that will use it to remotely connect to the Argos Sensor Service. Consequently the Sensor Configuration Tool only allows valid global IP/host addresses to be used when connecting.

The sensor options available to a sensor configuration is specified by the sensor profile as described in the previous section. In the Sensor Configuration Tool any values can be entered for the available options and it is up to the user to insert correct information. The sensor options is used by the Mobile Sensor Framework to configure the sensor before it starts collecting sensor data.

The last sensor configuration option specifies how fast data should be retrieved from a sensor. This information is specified as days, hours, minutes and seconds. The Mobile Sensor Framework converts this information into milliseconds and causes the thread assigned to retrieve sensor data to sleep for the specified interval between data retrievals. This means that if all fields are zero sensor data will be retrieved and distributed as quickly as possible.

The Sensor Configuration Tool also requires the user to specify additional information that is used by the Argos Sensor Service to deploy sensor configurations.

This information includes:

- How the Argos Sensor Service shall notify a device or computer about a new or modified configuration.
- Optional meta information describing the sensor

Currently the only notification method offered is SMS. This is because SMS messages are the only way handheld devices can be notified about something. However, if a version of the Mobile Sensor Framework was developed for regular computers a different notification mechanism would also be offered.

Optional meta information can be entered to allow the Argos Sensor Service to offer this information to Argos applications. In this prototype this information is limited to the *owner* of the sensor and the *location* of the sensor. For future development a more extensible approach should probably be implemented.

6.5.2 The Argos Sensor Service

The Argos Sensor Service takes care of storing and deploying sensor profiles and configurations. It is also responsible for receiving sensor data from the Mobile Sensor Framework and distributing this data to Argos applications that have registered to receive it. The Argos Sensor Service relies on the Argos Hibernate service discussed in section 4.4.1 to persistently store the profile and configuration XML documents. To simplify storage HyperJAXB discussed in section 4.5.2 was briefly considered to automatically generate hibernate mapping files for the JAXB objects used by the Argos Sensor Service. This would have been the easiest and most elegant approach to storing the XML documents, but unfortunately HyperJAXB could not be used due to reasons described in 4.5.2. The approach that was chosen instead was to store the XML documents as byte arrays directly in the database using Hibernate. This solution is not very elegant because it uses more storage space than necessary and really defeats the purpose of using Hibernate. However, it has the important advantage that changes to the XML schemas do not require any changes to how they are stored. This benefit greatly outweighs the drawbacks since the small number of documents to be handled makes storage space a non-issue.

To communicate with the Mobile Sensor Framework the Argos Sensor Service exposes three predefined Web Services. Predefined, in this case, means that their entry point and parameters should not change except for the host address. This allows the Mobile Sensor Framework to communicate by only knowing the host address of the computer running the Argos Sensor Service. The exposed Web Services are listed in figure 6.5.2.

Entrypoint	Method
http://ADDR:8080/axis/services/!!SensorService	getSensorConfiguration(String id);
http://ADDR:8080/axis/services/!!SensorProfileService	getSensorProtocolPlugin(String name, String type, String model, String manufacturer, String pluginid);
http://ADDR:8080/axis/services/!!SensorService	deliverData(byte[] packetData);

Like discussed in section 6.3 the Argos Sensor Service can receive sensor data both using an exposed Web Service and using the TCP/IP Service described in section 4.4.4. The port of the tcp server is specified in the sensor configuration, which makes it unnecessary to lock the implementation to a specific port. After the data is received the Argos Sensor Service interprets the XML and distributes the data to Argos applications that have registered to receive it.

6.5.3 The Mobile Sensor Framework

The Mobile Sensor Framework is implemented using the .NET Compact Framework for Windows Mobile 5.0 devices. Its main purpose is to simplify extraction of sensor data by using pluggable modules called *sensor protocol plugins*. Sensor protocol plugins are retrieved from the Argos Sensor Service along with configurations that describe operational parameters. Configurations are represented by XML documents that can be created by the Sensor Configuration Tool and sensor protocol plugins are .NET Compact Framework library files (.dll).

Since the Mobile Sensor Framework is running on a cell phone it can only be contacted using SMS. The Argos Sensor Service sends out special SMS messages when new or modified sensor configurations are available. The Mobile Sensor Framework uses a SMS handler to listen for these messages in order to intercept them before they are delivered to the user of the phone. The messages use a compact XML format that easily can be distinguished from other types of messages. An example of such a message can be seen in listing 6.4.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<announce id="12" ip="129.242.13.176" xmlns="sensorAnnouncement-1.0"/>
```

Listing 6.4: SMS notification

The SMS notification messages contain two pieces of information; An *id* that identifies the sensor configuration to be retrieved and the ip address of a computer running Argos. It is assumed that the Argos is running the Argos Sensor Service. As seen in figure 6.5.2 the Argos Sensor Service exposes a Web Service

that the Mobile Sensor Framework can use to retrieve the configuration. The Web Service must be supplied with the id from the notification in order to return the correct configuration.

The configuration specifies a sensor protocol plugin that can be used to collect data from the sensor. This plugin can be retrieved by using another Web Service exposed by the Argos Sensor Service. When the plugin, which more specifically is a .NET Compact Framework library file, has been retrieved it will be loaded dynamically by the Mobile Sensor Framework. An unfortunate limitation of the .NET Compact Framework is that library files cannot be unloaded or reloaded in any way. This means that plugins will only be fetched and loaded once while the Mobile Sensor Framework is running. Although hot deployment of plugins would be a nice feature, especially for testing, it is not very likely that you would want to change your plugin implementation and redeploy it while the Mobile Sensor Framework is running. Nonetheless, since this functionality is supported in the non-restricted version of the .NET Framework it is likely that the Compact Framework will support it sometime in the future.

The Mobile Sensor Framework provides two important pieces of functionality to sensor protocol plugins. First of all it provides a communication abstraction that allows plugins to interact with sensors without knowing or caring about how information is actually exchanged. That is, the plugins do not know if the sensor communication is done using bluetooth, IR or another type of protocol. Secondly, it provides functionality for delivering sensor data to its destination without plugins knowing or caring about where or how the data is sent.

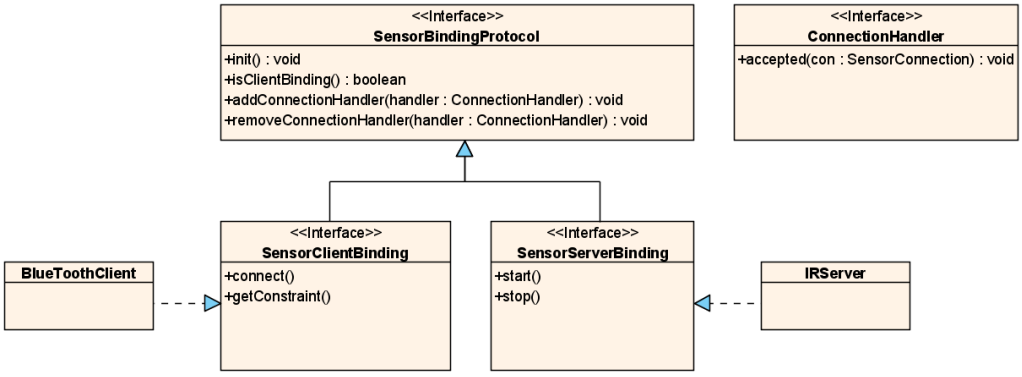

Figure 6.2: The Mobile Sensor Framework communication abstraction

The class diagram in figure 6.2 shows some of the interfaces that the sensor communication abstraction is made up of. Implementations of different communication protocols like bluetooth and IR must implement either the *SensorClientBinding* or *SensorServerBinding* interface. In many cases there will be two

implementations for each protocol; one for the server interface and one for the client interface. This distinction is necessary since it is not known who will initiate the connection; the handheld device or the sensor. Sensor protocol plugins will only work with instances of the client or server interface and are oblivious to the underlying protocol implementation. In order to communicate with a sensor they must register an object that implements the *ConnectionHandler* interface with the client/server binding object. After a connection is established the plugins receive a *SensorConnection* object that can be used to send and receive data. This *SensorConnection* interface, which is shown in figure 6.3 must be implemented by all communication protocols available to protocol plugins.

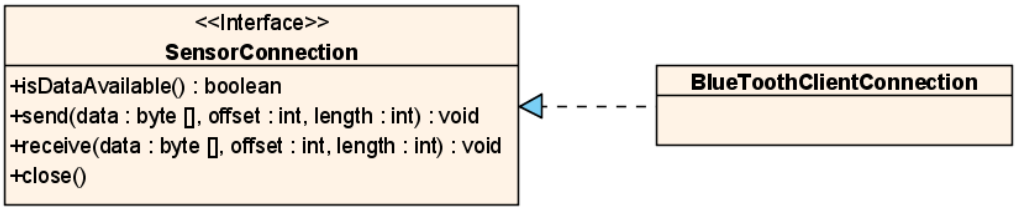

Figure 6.3: Sensor Connection

The Mobile Sensor Framework also provides functionality for distributing sensor data to remote receivers. Sensor protocol plugins use this functionality to deliver their measurement information every time they collect new data. Each sensor configuration specify how and where the sensor data should be delivered. This information is hidden from the sensor protocol plugins and is only used by the core of the Mobile Sensor Framework. Since the Argos Sensor Service can receive sensor data using either a Web Service or using a tcp/ip server the Mobile Sensor Framework has been implemented to support both of these methods.

6.6 Summary

This chapter has presented the implementation of a prototype system to collect and distribute sensor data to Argos applications. The implemented system consists of three main components that cooperate with each other. The Sensor Configuration Tool is a Java swing GUI application that enables users to create sensor configurations and sensor profiles. This tool interacts with the Argos Sensor Service, a system service in Argos, to store and deploy created sensor configurations and profiles. The Argos Sensor Service can deploy sensor configurations to handheld devices that are running the Mobile Sensor Framework. The Mobile Sensor Framework is implemented in the .NET Compact Framework and uses pluggable program modules, called sensor protocol plugins, received from the Argos Sensor Service to contact and collect sensor data from sensors. It also

takes care of sending the collected data to the Argos Sensor Service in order for this data to be distributed to Argos applications.

Chapter 7

Testing

The test chapter consists of two main sections. The first section will review the implementation to determine if the system functionality satisfies the requirements specified in chapter 3. The second section will demonstrate that the system works as intended by showing how an application developed for Argos can collect sensor data using the Argos Sensor Service.

7.1 Conformance Testing

The conformance testing was performed to check that different parts of the functionality works as expected. This testing was accomplished by observing the system components behavior when utilizing the use cases specified in the requirements chapter. The demonstration application that is described in the second main section also served as the basis for many of the tests that were performed. In these cases it is merely described how the demonstration application was used to test the functionality in question.

7.1.1 The Sensor Configuration Tool

Use case T-1: Connect To Argos

The requirements for this use-case specifies that the Sensor Configuration should allow a user to connect to a remotely running Argos and retrieve and display available sensor profiles that the Argos Sensor Service running on this Argos has stored. This use-case was tested by running the Argos Sensor Service on a Argos with a fixed number of sensor profiles and connecting to it with the Sensor Configuration Tool from a remote computer. In addition to the regular scenario a number of potential error situations were tested:

- Having zero sensor profiles in the Argos Sensor Service
- Inserting the wrong Argos address in connection settings of the Sensor Configuration Tool
- Terminating the connected Argos

All these situations were properly handled. The only exception is terminating the connected Argos which can still cause some minor glitches in the user interface. However, if the remote Argos is terminated the user of the Sensor Configuration Tool will be notified and can choose to reconnect.

Use case T-2: Create Sensor Profile

The Sensor Configuration Tool shall allow users to create new sensor profiles. The requirements for this use-case specifies that the sensor profile should be unique, that it should contain some mandatory items and that the profile shall be stored at the remote Argos that the tool is connected to. To test this use-case the Sensor Configuration Tool was connected to a remote Argos running the Argos Sensor Service and a new profile was created. The Sensor Configuration Tool allows all the mandatory items specified by requirement SCT-103 to be part of a new sensor profile. Uniqueness is based on the sensor profile identification fields and was tested by adding another profile with the same fields as the first one. This causes the Sensor Configuration Tool to notify the user that non-unique sensor profiles cannot be created. To test that the sensor profile was properly stored at the remote Argos the Sensor Configuration Tool was terminated, reopened and reconnected to the Argos. The reconnection caused the Sensor Configuration Tool to receive all stored sensor profiles and it could be observed that the newly created profile indeed was received. Consequently it must have been stored by the Argos Sensor Service.

Use case T-3: Edit Sensor Profile

The Sensor Configuration Tool shall allow users to edit existing sensor profiles. When edited the requirements specify that the Argos Sensor Service the tool is connected to shall update its persistent version of the profile. To test this functionality the tool was connected to a remote Argos running the Argos Sensor Service and an already existing profile was edited. To ensure that the Argos Sensor Service indeed had updated its persistent version of the profile the Sensor Configuration Tool was terminated, reopened and reconnected to the Argos. The reconnection caused the Sensor Configuration Tool to receive all stored sensor profiles and it could be observed that the edited profile had been properly updated.

Furthermore it was tested that sensor protocol plugins that were readded to a sensor profile were properly updated. A sensor profile must contain one or more sensor protocol plugins and when editing a profile you can delete a plugin to add a new plugin or to update the exiting plugin. To test that this actually worked as expected a sensor configuration was created based on the profile in question. Afterwards the configuration was deployed two times using the Sensor Configuration Tool to a mobile device. The first time the unedited version of the sensor profile was used and the second time the edited version was used. On the mobile device it was verified (by printing to screen) that the received sensor protocol plugin indeed had changed appropriately.

Use case T-4: Delete Sensor Profile

The Sensor Configuration Tool shall allow users to delete existing sensor profiles. When deleted the Argos Sensor Service the tool is connected to shall delete its persistent version of the profile. To test this use-case the tool was connected to a remote Argos running the Argos Sensor Service and an already existing profile was deleted. To ensure that the profile was actually deleted by the Argos Sensor Service the Sensor Configuration Tool was terminated, reopened and reconnected to Argos. The reconnection caused the Sensor Configuration Tool to receive all stored sensor profiles and it could be observed that the deleted profile was not among the received profiles.

Use case T-5: View Sensor Profile

The Sensor Configuration Tool shall allow users to view an existing profile. This use-case could be tested by just selecting a profile and clicking on the view button in the user interface. It worked as expected.

Use case T-6: Specify Sensor Identification

This functionality was tested when testing the creation of new sensor profiles. A sensor profile is identified by a fixed number of strings and the Sensor Configuration Tool makes sure that only unique profiles can be created.

Use case T-7: Configure Sensor Options

This functionality was tested when testing the creation of new sensor profiles. A sensor profile can contain a number of sensor options that specifies available configuration preferences for specific type of sensor. The Sensor Configuration Tool allows a profile to contain zero or more sensor options.

Use case T-8: Configure Sensor Interaction Descriptions

The Sensor Configuration Tool shall allows a user to add a number of sensor interaction descriptions to a sensor profile when the profile is created. The requirements for this use-case specify that sensor interaction descriptions *shall provide the information necessary to extract and interpret the measurements the sensor provides*. The sensor interaction descriptions ended up being called *sensor protocol plugins* as described in section 5.3.2 of chapter 5. The Sensor Configuration Tool allows zero or more sensor protocol plugins to be added to a sensor profile when the profile is created or edited. The actual adding functionality was tested when testing the creation of sensor profiles. Requirement SCT-117 specifies that no duplicate sensor protocol plugins can be added to a profile. A sensor protocol plugin is identified by its filename and when testing the creation of sensor profiles is was tested that a plugin with the same filename as an already added plugin could not be added.

Use case T-9, T-10, T-11: Create, Edit and Delete Sensor Configurations

These use-cases were relatively simple to test because the sensor configuration are not stored anywhere (as far as the Sensor Configuration Tool is concerned). Testing was performed by using the Sensor Configuration Tool GUI to create a new sensor configuration, editing it and afterwards deleting it. From the user interface it was obvious that this functionality worked as expected.

Use case T-12: Save Sensor Configuration

This use-case specifies that the Sensor Configuration Tool shall allow users to save sensor configurations that can be used outside the Sensor Configuration Tool. This functionality was implemented so that users can save sensor configurations as an XML file from the Sensor Configuration Tool. To test this feature a sensor configuration was created and afterwards saved as an XML file. It works as intended and the resulting XML file is verified by the Sensor Configuration Tool against an XML schema to be certain the format is correct.

Use case T-13: Configure Sensor Profile Options

When creating a sensor configuration the configuration can contain sensor options that may have been specified in the sensor profile the configuration is based on. The requirements for the use-case specifies that when the sensor options in a sensor profile changes all sensor configurations that use this profile shall be updated accordingly. This functionality was tested by trying to change the sensor options in a profile that was used by an existing sensor configuration. When an option was removed from the profile the option disappeared from the sensor configuration that used the profile. Similarly, when an option was added to

the profile the option appeared in the sensor configuration that used the profile. This was the expected result.

Use case T-14: Deploy Configuration

This use-case specifies that the Sensor Configuration Tool shall allow a user to deploy a sensor configuration to a mobile device. The Sensor Configuration Tool user interface has a deploy button that can be pushed when a sensor configuration is selected to deploy the configuration. When a configuration is deployed it is sent to the Argos Sensor Service, which notifies the device using the selected notification method (which is currently only SMS). The device then retrieves the configuration using a web service, retrieves the sensor protocol plugin specified by the configuration using a different web service and starts collecting sensor data using the plugin (unless otherwise specified by the configuration). The deployment functionality was tested by seeing if a mobile device would receive the configuration when deployed from the tool. All actions were logged to see if everything was performed correctly and it was verified that the mobile device received the configuration and the sensor protocol plugin.

7.1.2 The Argos Sensor Service

Many of the use-cases with accompanying requirements for the Argos Sensor Service has been tested when testing the Sensor Configuration Tool. Consequently all these use-cases have been left out in this section.

Use case A-6: Deliver Sensor Data

This use-case specifies that the Argos Sensor Service must provide a service that allows the Mobile Sensor Framework to deliver data to it. The Argos Sensor Service provides both a web service and a tcp/ip server to accomplish this task. To test this functionality sensor data was collected from a real sensor and distributed to the Argos Sensor Service. The test was performed using the GPS monitoring application described later in section 7.2. Using this application it was verified that sensor data could be properly delivered using both services.

Use case A-7: Receive Configuration Deployment Notifications

This use-case specifies that the Argos Sensor Service must make it possible for Argos applications to listen for configuration deployment notifications. That is, be notified when a configuration is deployed. The notification contains all the mandatory information specified in requirement ASS-112. Testing this functionality was done using the GPS monitoring application described later in section 7.2. The monitoring application is an Argos application that can receive GPS

sensor data using the Argos Sensor Service. By letting this application register itself with the Argos Sensor Service to receive deployment notifications and deploying test configurations from the Sensor Configuration Tool it was possible to test that this functionality works as intended.

Use case A-8: Receive sensor data

This use-case specifies that the Argos Sensor Service must make it possible for Argos applications to register to receive sensor data from a sensor referenced by a deployed sensor configuration. When they register they must supply the unique ID of the sensor configuration. The ID can be obtained when an Argos application receives a configuration deployment notification. This use-case was tested by using the GPS monitoring application described later in section 7.2 and it worked as expected.

Use case A-10, A-11: Getting sensor configurations and profiles

These use-cases specify that the Argos Sensor Service must make it possible for Argos application to retrieve sensor configurations and profiles for sensors that have been deployed. Both these use-cases was tested by using the GPS monitoring application described later in section 7.2. The functionality worked as expected.

7.1.3 The Mobile Sensor Framework

Similar to the Argos Sensor Service, many of the use-cases for the Mobile Sensor Service have been tested when testing the Sensor Configuration Tool. Furthermore, the requirements related to other mobile applications using the Mobile Sensor Service have been left out because there was no time to properly test them. However, these requirements also have a lower importance as they have been marked as *should* be implemented.

Use case M-2: Receive Sensor Data

This use-case specifies that the Mobile Sensor Framework shall be able to distribute sensor data to the Argos Sensor Service and that it should also be able to distribute it to mobile applications running on the device. The latter requirement has not been tested, but the former has been tested using the GPS monitoring application described later in section 7.2. This test also tested requirement MSF-104 which specified that the Mobile Sensor Framework shall properly constrain sensor data to be distributed in intervals as specified by sensor configurations. The last requirement for this use-case specifies that the Mobile Sensor Framework should be able to receive sensor data from multiple sensors simultaneously. Unfortunately this requirement could not be satisfied because of an unresolved bug.

7.2 A GPS monitoring application

The system developed in this thesis aims to make it easier to develop Argos applications that use measurements collected from remote sensors. To show that this is true a demonstration application was developed that uses the Argos Sensor Service to collect sensor measurements from the EMTAC CRUX II Wireless GPS Sensor that was described in section 6.4.1. The application developed is only supposed to receive and display the collected GPS location information.

7.2.1 NMEA Sentences

Like previously described, the EMTAC GPS Sensor is a bluetooth device that broadcasts a number of different NMEA sentences containing GPS information. The most important one of these sentences is the Global Positioning System Fix Data (GGA) sentence, which contains the latitude and longitude readings. The GGA NMEA sentence broadcasted looks like this:

```
$GPGGA,123519,4807.038,N,01131.000,E,1,08,0.9,545.4,M,46.9,M,,*47
```

and the sentence has the following format:

```
$GPGGA,hhmmss.ss,llll.ll,a,yyyyy.yy,a,x,xx,x.x,x.x,M,x.x,M,x.x,xxxx*hh
```

The different fields of the sentence is described by table D in appendix D.

In order for the demonstration application to receive GPS data from the Argos Sensor Service a sensor configuration for this sensor has to be created using the Sensor Configuration Tool. However, to be able to create a sensor configuration a sensor profile for the EMTAC GPS sensor first had to be defined. Filling out the necessary fields to create a EMTAC GPS Sensor profile is quickly accomplished using the Sensor Configuration Tool, but the profile also needs to be linked to a sensor protocol plugin that can interpret the GGA NMEA sentences broadcasted by the device.

7.2.2 EMTAC GPS Sensor Protocol Plugin

The sensor protocol plugin developed for the EMTAC GPS Sensor is responsible for parsing GGA NMEA sentences received from the sensor and converting each reading into an XML packet using the generic packet format defined in section 6.4. An example of such a packet can be seen in listing 6.1. A small code excerpt shown in listing 7.1 from the EMTAC GPS protocol plugin gives an overview of how the plugin interacts with the sensor.

```
public void getData(){
    /* Get data from sensor */
    while (!terminate){
        /* Read one character from stream */
        connection.receive(buf, 0, buf.Length);

        /* If start of NMEA sentence */
        if (buf[0] == '$'){
            start = true;
        }

        if (start){
            /* If not reached end of NMEA sentence */
            if (buf[0] != '\r')
                s += Encoding.ASCII.GetString(buf, 0, buf.Length);
            else{
                parseNmeaSentence(s, type);
                break;
            }
        }
    }
}
```

Listing 7.1: How the EMTAC GPS Sensor protocol plugin communicates

As seen in listing 7.1 the protocol plugin communicates using the stream abstraction provided by the Mobile Sensor Framework. The actual communication protocol used is specified by the sensor configuration and may be different even for sensors having the same sensor profile. The EMTAC GPS sensor we are using can only use bluetooth, which means 'bluetooth' must be selected as the sensor communication protocol in the sensor configuration.

The loop seen in listing 7.1 terminates when the plugin has received a complete NMEA sentence or if a timeout occurs. The Mobile Sensor Framework decides based on the sensor configuration how often this function should be called. Even though sensor protocol plugins run in their own threads, the threads are completely managed by the Mobile Sensor Framework who decides when and how often they should run.

7.2.3 The completed GPS monitoring application

The completed GPS monitoring application registers itself with the Argos Sensor Service to receive sensor packets from the EMTAC GPS Sensor. The sensor packets are represented by regular java objects because the Argos Sensor Service takes care of converting the XML to the corresponding JAXB objects. Although the GGA sensor packet contains many different fields the Argos GPS monitoring

application only extracts the GPS location information. This information is displayed in a simple GUI as seen in figure 7.1.

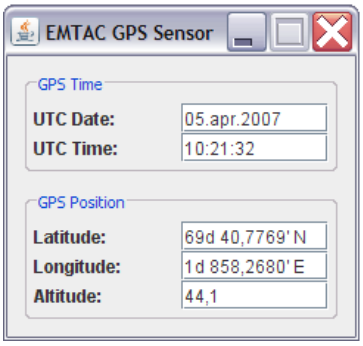

Figure 7.1: The Argos GPS Monitoring Application

Developing an Argos application to get the GPS sensor data without using the Sensor Configuration Tool, the Argos Sensor Service or the Mobile Sensor Framework would involve reinventing a lot of the functionality they already provide. You would, for instance, have to develop a mobile application that can interact with the EMTAC GPS Sensor using bluetooth and create functionality for sending the received information back to Argos. You would likely also have to create your own format to represent the information. Using the system created in this thesis all these problems are taken care of. Even though a sensor protocol plugin must be developed to interact with the EMTAC GPS Sensor it can be created with very little effort. This is especially true since it does not require you to know anything about bluetooth.

7.3 Summary

The tests outlined in this chapter have demonstrated that most of the requirements specified for the system were fulfilled and that the functionality works as expected. Furthermore, the GPS monitoring application that was developed has demonstrated that the system can be used with real sensors and real mobile devices.

Chapter 8

Evaluation

This chapter will evaluate the completed system in respect to the requirements specified in chapter 3 and determine if it satisfies the goals specified in the thesis problem definition. Furthermore, Argos will be evaluated in terms of how it was to use and develop functionality for.

8.1 Argos

The functionality developed in this thesis is specifically created for the Argos middleware platform. It is intended to simplify sensor data extraction so that Argos applications can receive and use sensor measurements with minimal amount of work. To extend Argos with this functionality a sensor system service was developed that can be plugged into the platform by applications that need it.

Developing Argos System Services is much like developing regular Argos applications. The only real difference is how they are loaded by the Argos Core and that they can subscribe to Argos Core events. The events lets a system service know what services and/or components are loaded, started and stopped. In previous Argos versions all the system functionality was tightly coupled with the Argos Core and you could not easily add new functionality. By having such a simple programming model it is very easy to learn both how to create Argos applications and how to extend the Argos functionality.

The biggest problem with developing functionality for Argos in this thesis was that the new pluggable system service architecture was being developed simultaneously. It took quite a while before the implementation was stable enough to be used and a lot of time was spent finding and correcting bugs that resided in this Argos functionality. On the bright side the close contact with the Argos

developer lead to a better understanding on how it was supposed to be used and also probably resolved bugs quicker than if they had been discovered when he was unavailable. This cooperation also probably made the Argos System Service functionality more robust since many of the errors encountered would not have been found through regular testing.

Among the advantages of the pluggable system service architecture is that it allows independent developers to create system services that easily can be shared. One example is the SMS Service, which was listed in section 4.4.2 in chapter 4. This system service was developed by a fellow student for his master thesis and was reused in the system developed for this thesis without any modifications. Hopefully the system service developed in this thesis will be as useful to future Argos developers.

8.2 Functional Evaluation

The functional evaluation of the system will be based on the requirements specified in chapter 3. However, due to the large number of requirements it will be focused on the requirements that were not supported, if any.

8.2.1 System Limitations

Due to the experimental nature of the system developed in this thesis there is naturally a few limitations to its use. One of the biggest is probably that no interoperable way was found to describe generically how to do sensor interaction and data extraction. During the project inception it was believed that it would be possible to describe this interaction in an XML-type of language that could be reused across different platforms. This would have allowed sensor protocol plugins to be developed once to be run on Mobile Sensor Frameworks written for different, incompatible mobile device platforms. Unfortunately the XML approach was dropped because it did not seem possible to create a format that would be compatible with all types of sensors.

A different limitation or challenge is how users should share sensor profiles with each other in order for them to be reused. In this thesis Argos acts as a central repository for sensor profiles, but if everyone is running their own repository reuse will be limited since everyone will have use their own profile library. This is a difficult problem that it seems only standards can solve. While everyone should probably have their own repository server, they must have some way to find and add new sensor profiles and protocol plugins provided by others. If a standardized sensor profile format and API for sensor protocol plugins was defined it is possible to envision that companies developing sensors could provide

sensor profiles and protocol plugins to customers. To create such a standard and get companies to agree to support it is an entirely different matter.

The last limitation is related to the sensor communication protocol abstraction that the Mobile Sensor Framework provides to sensor protocol plugins. The abstraction was created to hide the real protocol used to interact with sensors by providing a standard communication API. However, since only a bluetooth implementation was developed it is possible that adding new protocols will require modifications to how the abstraction is implemented. The API should not depend on any bluetooth specific functionality, but it cannot be guaranteed to be completely independent before more protocols are added.

8.2.2 Missing functionality

Some pieces of functionality got left out simply because there was not enough time to implement them. Most of these features were expected to make it into the final system, but were discarded because other features were considered more important. Users using the Sensor Configuration Tool will probably miss the ability to save sensor configurations in such a way that they will be available next time they open the program. This feature was not left out because it is not useful, but because it was not vital functionality to demonstrate the usefulness of the whole system. At some point in the development process it was also planned to support both push and pull strategies to receive sensor data from the Mobile Sensor Framework. This plan was dropped in favor for just arguing why it would be best to support a push approach. A pull approach could still prove useful and can potentially be added in the future. Finally, the Mobile Sensor Framework does not take into consideration that the handheld device can be turned off and back on. This means that it will not remember sensor configurations it has received and restart its sensor interaction and distribution of sensor data. This functionality would be essential in order for this framework to be used in a real setting, but requires more planning and development.

8.2.3 Sensor Configuration Tool

The main purpose for the Sensor Configuration Tool is to provide a user interface to create sensor profiles and configurations. From user input it creates corresponding configuration and profile XML documents that conform to XML schemas provided by the Argos Sensor Framework. It satisfies all requirements specified for it in the requirements chapter, but the functionality it provides still have some minor flaws that was not fixed. The flaws are mostly cosmetic in that they only effect how the user interface works. It is, for instance, possible to

lockup the user interface so the application must be restarted, but these errors were deemed unimportant for the system as a whole.

8.2.4 Argos Sensor Framework

The Argos Sensor Framework ended up being the least complex of the system components developed. However, it is possible to look at the Sensor Configuration Tool and the Argos Sensor Framework as a single application where the former provides the view/presentation and the latter provides the logic/control. The Argos Sensor Framework's second purpose is to interact with the Mobile Sensor Framework in order to receive sensor data from remote sensor distribute this data to registered Argos applications. Furthermore it acts as a mediator for providing sensor configurations and sensor protocol plugins to the Mobile Sensor Framework. It notifies devices running the Mobile Sensor Framework when new or modified sensor configurations are received and exposes web services that can be used to retrieve configurations. This system component fulfills the second primary goal stated in the thesis problem definition in chapter 1 and conforms to all requirements specified.

8.2.5 Mobile Sensor Framework

The Mobile Sensor Framework fulfills the first primary goal stated in the thesis problem definition. It stated that a sensor framework should be developed for handheld devices that takes care of *accessing, retrieving and distributing sensor data*. The Mobile Sensor Framework is merely a library that does not do anything useful by itself. It contains generic functionality for accessing sensors and provides an environment for running sensor protocol plugins. Sensor protocol plugins are small programs that contain only sensor specific functionality. They can take advantage of the generic functionality provided by the Mobile Sensor Framework to easily collect and distribute sensor data. Since most of the complex functionality is provided by the Mobile Sensor Framework the sensor protocol plugins can be very compact. Their size will, however, depend on the complexity of the sensor they are written for. A sensor that has many measurements and a complicated interaction protocol will mean that the sensor protocol plugin must contain more logic to interact with it.

The features provided by the Mobile Sensor Framework is intended to capture all generic functionality needed by sensor protocol plugins. This means that the plugins:

- Do not need to contain logic to save or keep track of configuration settings
- Do not need to contain logic to connect to sensors

- Can use a uniform way to interact with all sensors (hides protocols like bluetooth, IR, WLAN, etc)
- Does not have to worry about how often or when they should get sensor data
- Do not need to know how or where the sensor data should be sent after collected

The Mobile Sensor Framework takes care of retrieving sensor configurations from the Argos Sensor Service without the sensor protocol plugins having to know or care about it. The sensor protocol plugins receive configuration objects from the Mobile Sensor Framework that contain all the configuration parameters they need when they are initialized. This way they do not need to save configuration settings or know that the settings are actually described in an XML document.

Among the features provided by the Mobile Sensor Framework the sensor communication abstraction is the most time saving when developing sensor protocol plugins. It provides a uniform API for sensor protocol plugins to connect to and interact with sensors effectively hiding protocols like bluetooth or IR. All configuration parameters related to the communication protocol are extracted from the sensor configuration before any settings are given to the plugins. When the plugins are initialized they just receive a binding object that has been preconfigured with the settings from the sensor configuration. The binding object exposes part of the communication abstraction API and allows a sensor protocol plugin to immediately start interacting with a sensor without specifying any more information.

Sensor protocol plugins are scheduled by the Mobile Sensor Framework. This means that the framework decides when and how often they should run. Developers must take this into consideration when developing plugins to avoid using blocking operations. Information on how often a plugin should run is specified in the sensor configuration and is only available to the Mobile Sensor Framework. It can be debated if it makes the plugins easier to develop since they must use non-blocking operations, but it ensures that the scheduling functionality is practically free of any bugs that otherwise could have been introduced by plugins.

The last piece of generic functionality provided by the Mobile Sensor Framework is sensor data distribution. Sensor protocol plugins need to assemble XML sensor packets that are appropriate for the sensor data they collect, but can otherwise just use the functionality provided by the Mobile Sensor Framework to distribute the data to remote receivers. This further simplifies the plugins and allows them to use an efficient delivery solution with minimum amount of fuzz.

The Mobile Sensor Framework satisfies most of the requirements specified for it in chapter 3, but requirement MSF-105 was not supported. This requirement indicates that the Mobile Sensor Framework should be able to run multiple sensor protocol plugins simultaneously to receive and distribute sensor data from multiple sensors. The Mobile Sensor Framework was developed with the intention to have this feature, but an unresolved bug caused it not to function properly. The upside is that once the bug gets resolved the Mobile Sensor Framework satisfies all requirements that were specified.

8.2.6 Comparison with Related Systems

The CommonSense ToolKit

In section 2.2 of the related work chapter it was described how the CommonSense ToolKit tries to deal with sensor interaction in a uniform fashion. This system uses a single XML document to specify both communication protocol settings and the sensor data format. However, as described in related work, the CommonSense ToolKit seems to assume that sensors will provide sensor data in a binary format and does not describe any further sensor protocol interaction. The system developed in this thesis recognizes these limitations and does not assume that a sensor will provide a binary protocol in order to work with as many sensors as possible. In our system a module in a specific programming language is used instead of an XML document to describe sensor interaction. This approach effectively sacrifices interoperability in order to support more sensors. The benefit of this approach can be demonstrated with the EMTAC GPS Sensor. The CommonSense ToolKit would not be able to communicate with this sensor even though the sensor does provide a binary protocol. The reason is that the binary protocol specifications are not publicly available. The system developed in this thesis can use the ascii format instead which it is possible to interpret because it follows the NMEA standard.

SensorML

SensorML was discussed in the related work chapter as a way to describe different types of sensors. It is basically a XML language specification that is created specifically for the sensor domain and relates to how sensor profiles are specified in this thesis. SensorML does not describe any information associated with protocols or data formats, but provides for instance all details concerning the measurements a sensor can supply. The system created in this thesis borrowed some of the concepts for describing sensor profiles from SensorML, but is otherwise very simple in comparison. One advantage of SensorML, which this system lacks, is standardized measurement units and phenomenon descriptions.

SensorML provides a large number of URIs to reference common units and phenomenons. This allows different types of sensors to have SensorML profiles where this type of information can be compared. Such information would be very useful if Argos applications are going to reason about sensor data collected from different sensor sources. In the future it would be an advantage to incorporate these units and phenomenon descriptions into the sensor profile format defined for this system.

8.3 Non-Functional Evaluation

The non-functional evaluation will concentrate on the non-functional requirements specified in chapter 3. The focus will mainly be on the systems flexibility, extensibility and interoperability.

8.3.1 Flexibility

Although flexibility is not specifically mentioned in any of the requirements for the system, the system is very flexible in how it can be used. This flexibility is mostly related to how sensor configurations can be deployed and created. The primary way to create and deploy sensor configurations is through using the Sensor Configuration Tool, but it is not the only way. Sensor configurations can be bundled with and deployed by Argos applications that contact the Argos Sensor Framework directly. The configurations themselves can be created manually by hand, but it is recommended to create them by using the Sensor Configuration Tool. The Mobile Sensor Framework primarily receives sensor configurations from the Argos Sensor Service, but it can also receive them from applications using the framework on the mobile devices. When receiving it from an application on the device it is assumed that the application has a sensor configuration bundled when it is installed.

8.3.2 Loose-coupling and interoperability

The three main system components developed are loosely-coupled in most respects. The Sensor Configuration Tool represents the user interface for the Argos Sensor Service, but is contained in a separate independent GUI application. This makes the GUI functionality loosely-coupled from the logic and can be easily replaced. A new GUI application could be developed that uses the API that the Argos Sensor Service provides to change the user interface appearance completely. However, these two system components communicate using JMX, which is based on RMI. This choice of communication ties these components to Java platform, which means that it would not be so easy to replace one of these components with a component created for a different platform. However, in this case the

simplified communication of JMX outweighs the benefits of having interoperable system components.

The Argos Sensor Service and the Mobile Sensor Framework are loosely-coupled and almost completely interoperable. There is nothing that makes the Argos Sensor Service and the Mobile Sensor Framework dependent on each other and all information is contained in XML documents and exchanged using Web Services. This satisfies requirement NFR-107 from the non-functional requirements section. The only exception is sending and receiving sensor data using tcp/ip. However, since tcp/ip is available on all platforms and the protocol used is very simple, it would take very little effort to replicate it in any programming language in a few lines of code. Interoperability is an important requirement for these components because there exists a number of different mobile platforms that could run the Mobile Sensor Framework. Developing the Mobile Sensor Framework another one of these platforms should not require any changes to the Argos Sensor Service.

8.3.3 Extensibility

The extensibility requirements specified in chapter 3 tries to ensure that it will be easy to accommodate changes to the system in the future. Such changes are likely to affect sensor configurations and sensor profiles since new types of sensors might reveal limitations in the current formats. For this reason the system components have been developed to depend as little as possible on the current formats so that they can be changed with minimal effort. The XML formats themselves have also been created using very extensible design patterns in order for changes to affect as few parts of the system as possible. Only the Sensor Configuration Tool is relatively tightly-coupled with the sensor profile and configuration formats. The reason is simply that it is a tool for creating documents that use those formats, which made it difficult to make them independent of each other.

One motivation for creating a communication abstraction for the Mobile Sensor Framework was to make sure that it can be extended with new sensor communication protocols without requiring changes to existing sensor protocol plugins. The plugins just continue to use the same API while a different communication protocol can be used underneath without their knowledge. Although the Mobile Sensor Framework currently only supports bluetooth, Infrared communication can be added with only minor changes to the framework.

8.4 Future Work

The system created in this thesis has numerous opportunities for future improvement, but some of these opportunities really depend on how you want to use the

system. This section provides some ideas to how the system can be improved both on a small and a large scale.

On a small scale there are many changes that can be made. Sensor configurations that have been previously created should be available when reopening the Sensor Configuration Tool. This feature was listed as missing functionality and did not get implemented due to time constraints. The Mobile Sensor Framework can provide additional features to sensor protocol plugins. One such feature could be encryption of sensor data. NST would probably welcome such a feature because patient data is considered very confidential and should not be transmitted insecurely. The Mobile Sensor Framework could also be extended with more connection protocols to be able to interact with, for instance, sensors that use IR for communication.

8.4.1 Transformations

One of the larger scale improvements is to let the Argos Sensor Service transform received sensor data into RDF data and have an option to store it in a RDF triplet repository like Sesame[1]. An RDF vocabulary could be created for each sensor profile through the Sensor Configuration Tool with additional information on how the transformation should be accomplished. This would benefit Argos applications by giving them a much better way to query sensor data since the RDF data can contain structured semantic information.

8.4.2 Discovery

The Sensor Configuration Tool does not currently provide any help with discovering sensors. For some types of communication protocols, like bluetooth, it would be possible to let the user of the Sensor Configuration Tool choose the sensor to use based on a list of discovered sensors. Such a feature would be relatively easy to implement using bluetooth since the list would just consist of discovered bluetooth devices. Additional options for filtering could also be provided.

8.4.3 Monitoring

It is possible to extend the functionality provided by the Sensor Configuration Tool far beyond what it is currently capable of. Instead of using it as just a configuration tool, it could be transformed into a complete configuration and monitoring tool for all devices and sensors associated with a Argos Sensor Service. Currently you can create sensor configurations for multiple sensors, but there is no indication if these sensors are connected to the same device. It would be

[1] http://www.openrdf.org/

better to have device configurations where each device can have multiple sensor configurations. The tool could then provide information on the status of devices and sensors based on feedback received by the Argos Sensor Service from the Mobile Sensor Framework. It would be possible to reconfigure both sensors and devices remotely to change their behavior and automatic notification mechanisms can be setup to react to status changes. From these ideas it is easy to imagine the Sensor Configuration Tool as more of a general purpose monitoring tool for mobile devices that can control more than just sensors. However this again depends on what type of functionality that is desired.

8.5 Summary

This chapter has provided an evaluation of the system developed in this thesis. The evaluation has tried to unveil both the advantages and disadvantages of the final system and shown how the different requirements specified have been fulfilled. Additionally ideas for future work has been provided that can be used for future development.

Chapter 9

Conclusion

This thesis has designed and implemented a system to simplify sensor data extraction for applications running on the Argos middleware platform. Many applications, especially context-sensitive ones, rely on measurements collected from remote sensors in order to accomplish their goals. Such applications are usually just interested in the actual measurement data and do not want to concern themselves with the complexities of sensor data formats, communication protocols and configuration settings. However, due to the lack of tools to simplify sensor interaction most applications choose to reinvent the wheel by developing all functionality needed themselves. This approach works, but increases development time and misses the great benefits that can be reaped from robust and reusable software libraries.

The system developed in this thesis solves these problems by providing a Sensor Service to Argos applications that hides the details of how sensor data is collected. Based on the problem scenario defined by NST a mobile application framework was developed to do the actual sensor interaction. Its purpose is to collect sensor data from sensors and send this data to Argos. One of the most difficult problems to solve was how this framework should work with all types of sensors when sensors have sensor specific issues that cannot be solved by a generic library. The solution was to separate common sensor functionality from functionality tied to each individual sensor. This was accomplished by allowing the mobile sensor framework to be extended with tiny application plugins containing sensor specific functionality. These plugins can be loaded on-the-fly to access new sensors without any modifications. To maximize reuse the system shares all sensor plugins within a single Argos so that they can be reused by any Argos application.

Although the system developed makes it easier for Argos applications to receive

sensor data it has plenty of room for improvement. Its main limitation is that the sensor plugins are tied to the programming language of the framework. Ideally they should be described in a interoperable markup language like XML which could be interpreted by the framework, but due to the large differences between sensors it did not seem possible to create such a markup language format.

The system created in this thesis is only a prototype used to illuminate issues related to sensor interaction and data extraction. Some of its features may not be suitable for production systems, but are still valuable in a research setting. NST has decided to continue development of this system to create a more comprehensive prototype that is even more tailored to their own goals and usage scenarios.

Bibliography

[1] Software development kit for coaguchek xs. Technical report, Roche Diagnostics, 2005. `http://www.coaguchek.se`. [cited at p. 12]

[2] Opengis geography markup language(gml) implementation specification, 2007. `http://www.opengeospatial.org/standards/gml`. [cited at p. 15]

[3] Roger Allan. The future of sensors. *ELECTRONIC DESIGN*, 2004. `http://www.elecdesign.com/Articles/Index.cfm?AD=1&ArticleID=8326`. [cited at p. 1]

[4] Trevor Armstrong, Cristiana Amza, Olivier Trescases, and Eyal de Lara. Efficient and transparent dynamic content updates for mobile clients. *MobiSys '06*, 2006. [cited at p. 20]

[5] I. Jacobson Booch, G. and J. Rumbaugh. *The Unified Modeling Language User Guide*. Addison-Wesley, 1999. [cited at p. 24]

[6] Mike Botts and Alexandre Robin. Opengis sensor model language (sensorml) implementation specification. Technical report, The Open Geospatical Consortium (OpenGIS), 2007. `http://vast.uah.edu/SensorML/`. [cited at p. 15]

[7] M. Cokus and D. Winkowski. Xml sizing and compression study for military wireless data. *Proceedings of the XML 2002 Conference*, 2002. [cited at p. 13, 20]

[8] Mani Srivastava David Culler, Deborah Estrin. Overview of wireless sensor networks. *IEEE Computer*, 2004. [cited at p. 18]

[9] Omoju A. Thomas David J. Russomanno, Cartik R. Kothari. Building a sensor ontology: A practical approach leveraging iso and ogc models. *The 2005 International Conference on Artificial Intelligence*, 2005. [cited at p. 17, 18]

[10] Dan Peder Eriksen. Design and implementation of the second generation of apms middleware, 2006. University of Tromsø, Norway. [cited at p. 2, 43]

[11] Van Laerhoven K., Berchtold M., and Gellersen H.-W. Accessing and abstracting sensor data for pervasive prototyping and development, 2005. [cited at p. 11]

[12] Mike Beedle Ken Schwaber. *Agile Software Development with Scrum*. Prentice Hall, 2002. [cited at p. 3]

[13] V. Kumar. Wireless communications - beyond 3g. 2001. [cited at p. 19]

[14] Ruth Malan and Dana Bredemeyern. Defining non-functional requirements. Technical report, BREDEMEYER CONSULTING, 2001. http://www.bredemeyer.com/pdf_files/NonFunctReq.PDF. [cited at p. 37, 38]

[15] A. Mani and A. Nagarajan. Understanding quality of service for web services. http://www-106.ibm.com/developerworks/library/ws-quality.html. [cited at p. 13]

[16] Frank Manola and Eric Miller. Rdf primer. Technical report, World Wide Web Consortium (W3C), 2004. http://www.w3.org/TR/rdf-primer/. [cited at p. 13]

[17] Sun MicroSystems. Jave enterprise edition faq. http://java.sun.com/javaee/overview/faq/javaee_faq.jsp. [cited at p. 42]

[18] William T.C Kramer Nathan Ota. Tinyml: Meta-data for wireless networks. 2003. [cited at p. 18]

[19] Sasank Reddy. Project report: Sensor data stream protocol. *Association for Computing Machinery (ACM)*, 2006. [cited at p. 9, 18]

[20] Darleen Sadoski and Santiago Comella-Dorda. Three tier software architectures. Technical report, Carnegie Mellon: Software Engineering Institute, 2000. http://www.sei.cmu.edu/str/descriptions/threetier_body.html. [cited at p. 52]

[21] M. Satyanarayanan. Fundamental challenges in mobile computing. *Principles of Distributed Computing (PODC '96)*, 1996. [cited at p. 19]

[22] M. Tian, T. Voigta, T. Naumowicz, H. Ritter, and J. Schiller. Performance considerations for mobile web services. *Elsevier Computer Comm. J.*, 27, 2004. [cited at p. 63]

[23] Cecilie Daae Tor Claudi, John G. Cooper. Er diabetesbehandling for vanskelig for allmennpraktikeren alene? *Tidsskrift for Den Norske Lgeforening*, 2000. http://www.tidsskriftet.no/pls/lts/PA_LT.VisSeksjon?vp_SEKS_ID=17616. [cited at p. 5]

[24] Sitalakshmiy Venkatraman. Mobile computing models are they meeting the mobile computing challenges? *Association for Computing Machinery New Zealand Bulletin*, 2005. [cited at p. 19]

Appendices

Appendix A

Appendix A contains visualizations of the three different XML schemas that have been developed in this thesis. These schemas define the structure of XML packets, sensor configurations and sensor profiles.

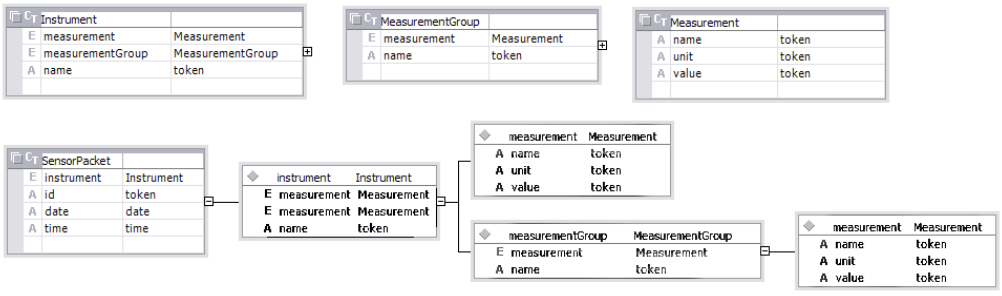

Figure A.1: A visualization of the sensor packet XML schema

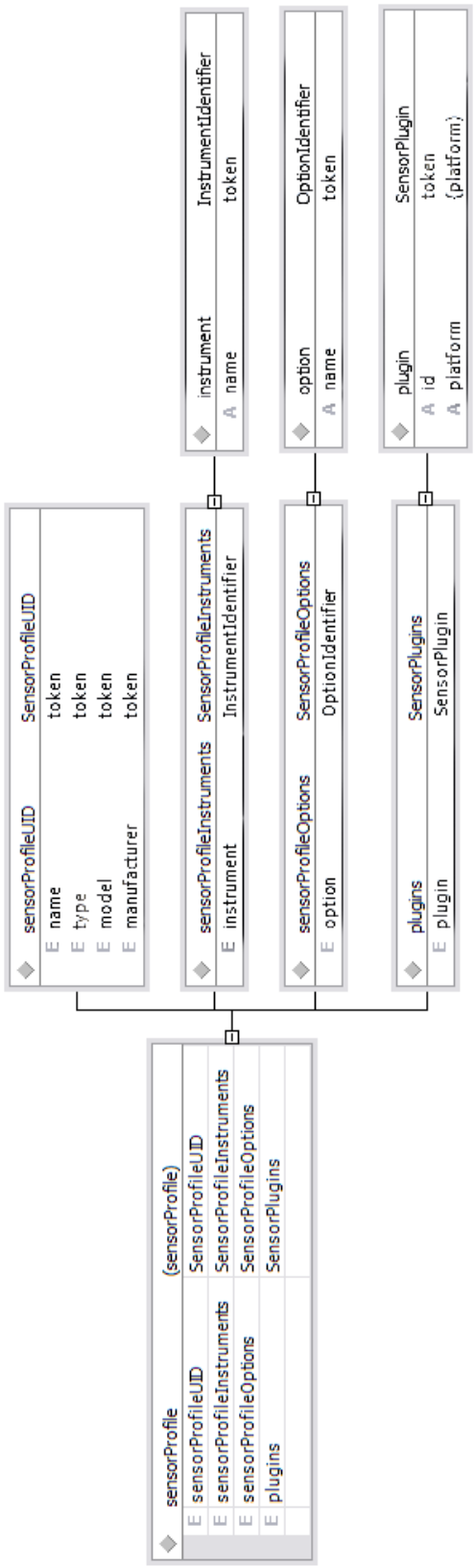

Figure A.2: A visualization of the sensor profile XML schema

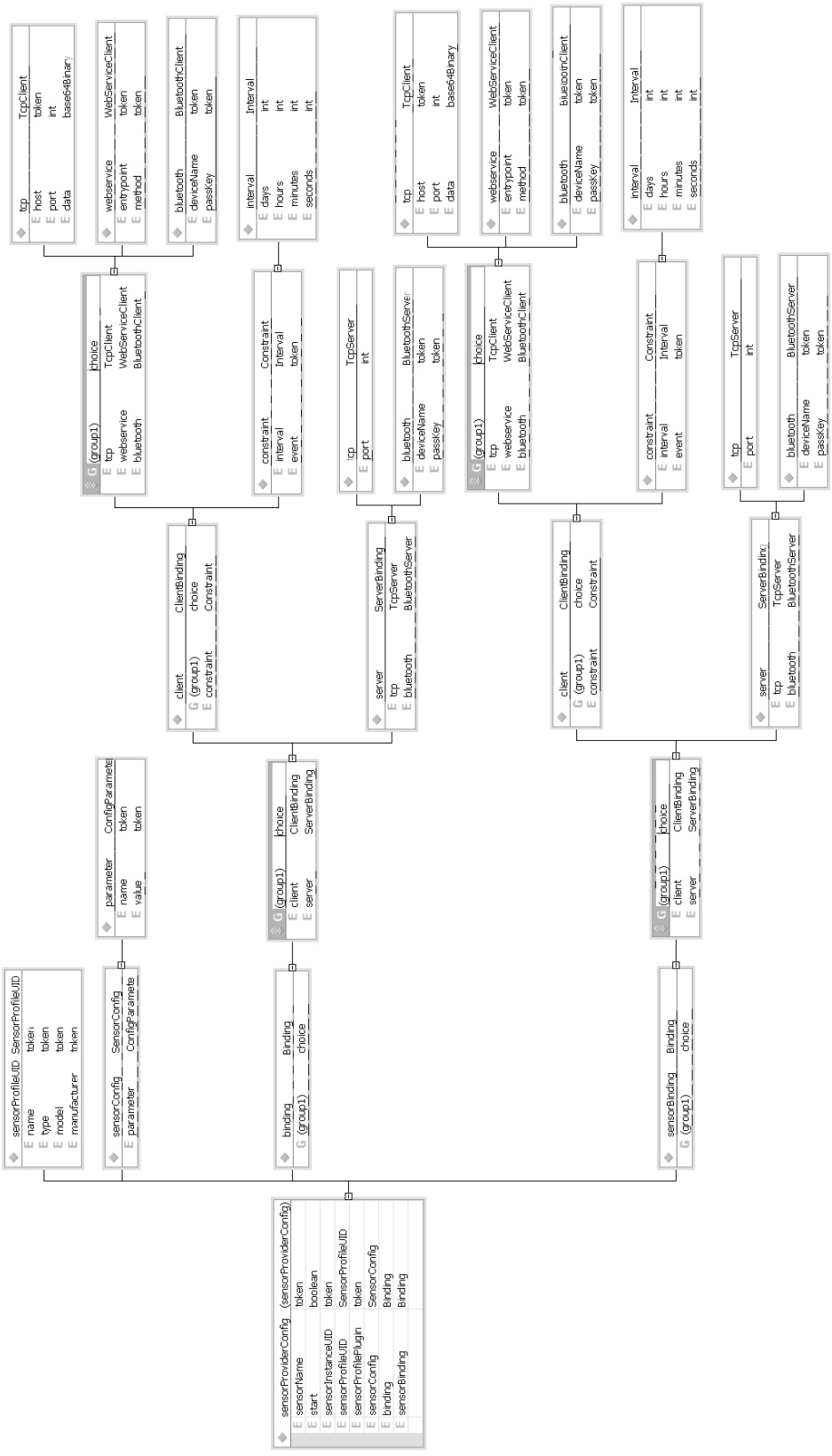

Figure A.3: A visualization of the sensor configuration XML schema

Appendix B

Appendix B contains complete versions of different XML examples that have been shown in the Implementation chapter.

```
<?xml version="1.0"?>
<sensorPacket id="21323423" date="2007-02-15" time="12:04:10"
xmlns="http://argos.cs.uit.no/sensors/packet-1.0">

<instrument name="GGA">
   <measurement name="Time" unit="time" value="170834"/>
   <measurement name="FixQuality" unit="int" value="1"/>
   <measurement name="Satellites" unit="int" value="05"/>
   <measurement name="HDOP" unit="double" value="1.5"/>
   <measurement name="Altitude" unit="meters" value="1"/>
   <measurement name="GeoidHeight" unit="meters" value="-34.0"/>

   <measurementGroup name="Latitude">
      <measurement name="Degrees" unit="degrees" value="41"/>
      <measurement name="Minutes" unit="minutes" value="24"/>
      <measurement name="Seconds" unit="seconds" value="54"/>
      <measurement name="Direction" unit="direction" value="north"/>
   </measurementGroup>

   <measurementGroup name="Longitude">
      <measurement name="Degrees" unit="degrees" value="81"/>
      <measurement name="Minutes" unit="minutes" value="51"/>
      <measurement name="Seconds" unit="seconds" value="41"/>
      <measurement name="Direction" unit="direction" value="west"/>
   </measurementGroup>
</instrument>

   <instrument name="GSV">
      ...
   </instrument>
</sensorPacket>
```


Listing B.1:]Generic Sensor Data Format Example [EMTAC GPS Sensor]

```
<outputs>
  <OutputList>
    <output name="weatherMeasurements">
      <swe:DataGroup>

        <swe:component name="time">
          <swe:Time
            definition="urn:ogc:def:phenomenon:time"
            uom="urn:ogc:def:unit:iso8601"/>
        </swe:component>

        <swe:component name="temperature">
          <swe:Quantity
            definition="urn:ogc:def:phenomenon:temperature"
            uom="urn:ogc:def:unit:celsius"/>
        </swe:component>
-
        <swe:component name="barometricPressure">
          <swe:Quantity
            definition="urn:ogc:def:phenomenon:pressure"
            uom="urn:ogc:def:unit:bar"/>
        </swe:component>
-
        <swe:component name="windSpeed">
          <swe:Quantity
            definition="urn:ogc:def:phenomenon:windSpeed"
            uom="urn:ogc:def:unit:meterPerSecond"/>
        </swe:component>
-
        <swe:component name="windDirection">
          <swe:Quantity
            definition="urn:ogc:def:phenomenon:windDirection"
            uom="urn:ogc:def:unit:degree"/>
        </swe:component>
-
        <swe:component name="rainFall">
          <swe:Quantity
            definition="urn:ogc:def:phenomenon:rainfall"
            uom="urn:ogc:def:unit:meter"/>
        </swe:component>
      </swe:DataGroup>

    </output>
  </OutputList>
</outputs>
```

Listing B.2:]Davis Weather Station Measurements [SensorML]

Appendix C

Appendix C contains screen shots taken of the Sensor Configuration Tool. Only the most important parts of the user interface have been included. It shows how the main screen looks along with how sensor configurations and profiles are created and edited.

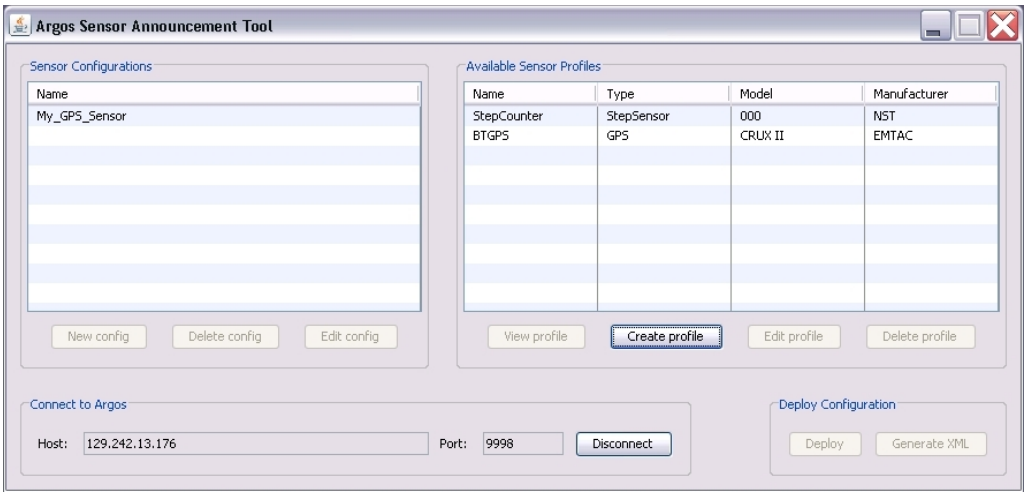

Figure C.1: The main window of the Sensor Configuration Tool

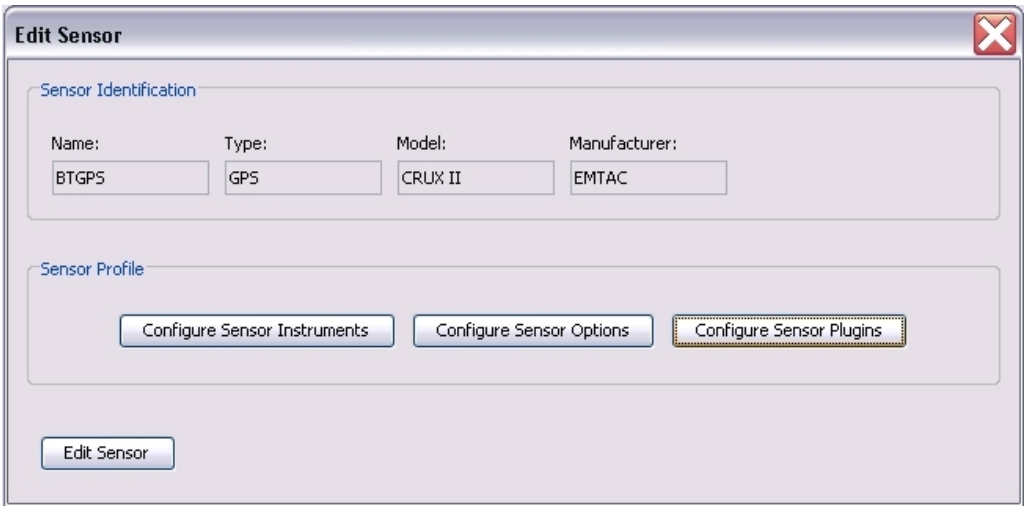

Figure C.2: Editing a sensor profile

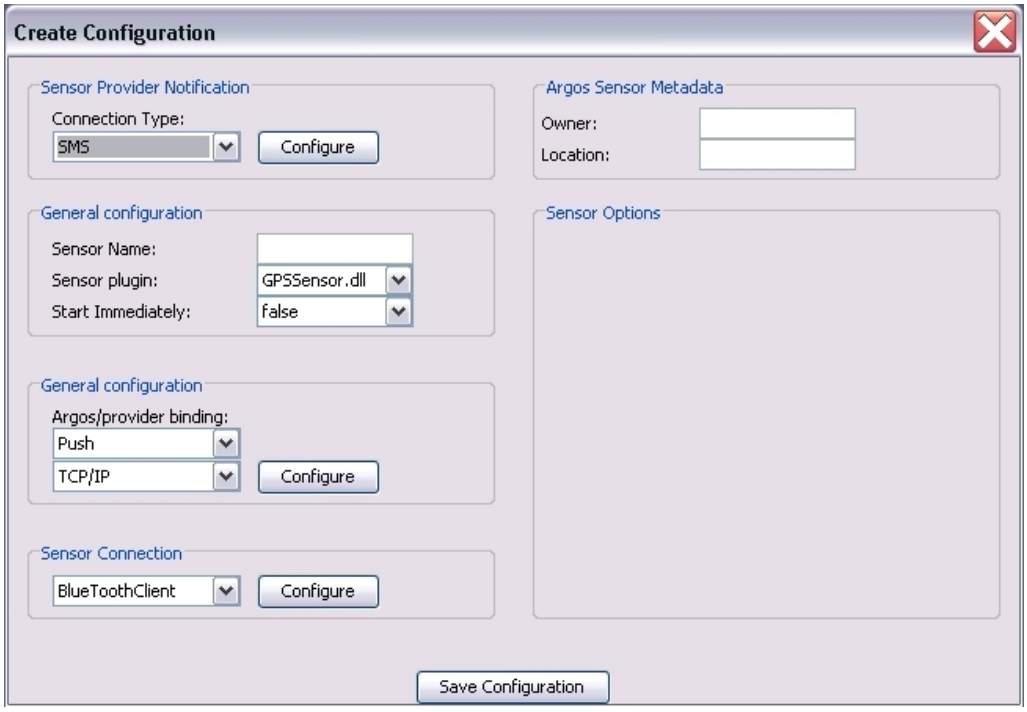

Figure C.3: Configuring a new sensor configuration

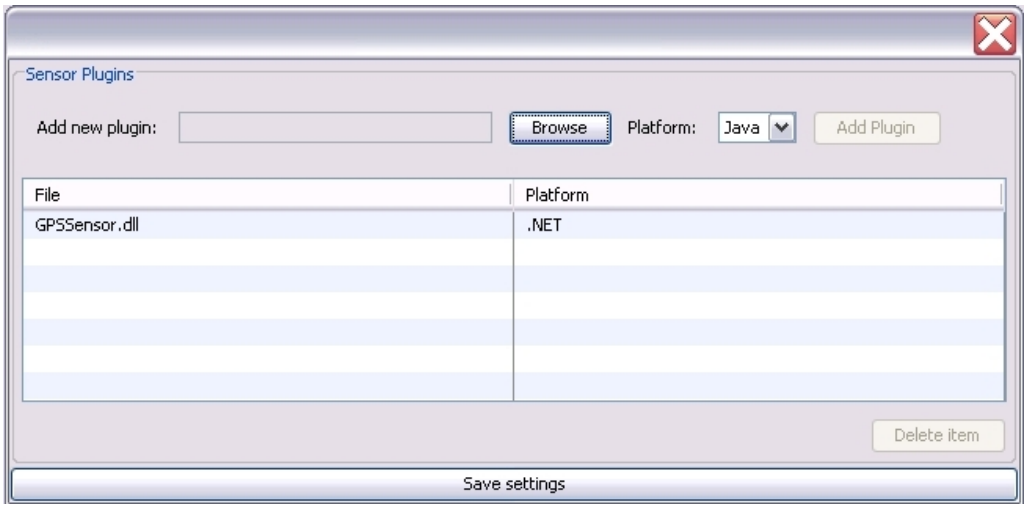

Figure C.4: Adding a sensor protocol plugin to a sensor profile

Appendix D

Appendix C contains more complete information on the GGA NMEA sentence interpreted by the GPS monitoring application described in 7.2 of chapter 7.

Field	Description
01	Sentence Identifier
02	Time (UTC) of Position
03	Latitude
04	Latitude Direction (north or south)
05	Longitude
06	Longitude Direction (east or west)
07	GPS quality indicator (0=Invalid, 1=GPS fix, 2=DGPS fix)
08	Number of satellites in use
09	Horizontal Dilution of Position (HDOP)
10	Antenna altitude above/below mean sea level
11	Meters (Antenna height unit)
12	Height of geoid above WGS84 ellipsoid
13	Meters (Units of geoidal separation)
13	Age in seconds since last update from diff. reference station
14	DGPS station ID
15	Checksum

Figure D.1: GGA Sentence Fields

CD-ROM

The CD-ROM contains this PDF document, all source files and everything need to get the system up and running. More specifically it contains the following items:

Thesis.pdf
This document.

Argos
This folder contains a compiled version of Argos that has been confirmed to work with the Argos Sensor Service.

BangBang SensorFramework
This folder contains the Argos Sensor Service as a Eclipse project.

SensorFrameworkDemonstrator
This folder contains the demonstration Argos application as a Eclipse project. It receives GPS measurements from the EMTAC GPS Sensor.

SensorConfigurationTool
This folder contains the Sensor Configuration Tool Eclipse project that can be used to create new sensor profiles and configurations.

SCTPlugins
This folder contains a plugin Eclipse project that extend the functionality of the Sensor Configuration Tool. If new notification mechanisms or sensor communication protocols are needed new plugins can be created in this project.

SensorXML
This folder contains the SensorXML Eclipse project. This project contains XML related functionality that is shared between most of the other projects.

MobileSensorFramework
This folder contains the Mobile Sensor Framework as a Visual Studio project. It also includes two Sensor Protocol plugin examples. One for the EMTAC GPS Sensor and one for a preliminary version of the NST Step Sensor.

Readme.txt
Consult this readme file for more detailed information.