NTNU
Innovation and Creativity

Context-Aware Services in Aquaculture

FiFaMoS - Fish Farm Monitoring System

Jon Arne Grødal
Frank Gjervik Paaske

Master of Science in Communication Technology

Submission date:	June 2006
Supervisor:	Per-Oddvar Osland, ITEM
Co-supervisor:	Frode Flægstad, Telenor R&D

Norwegian University of Science and Technology
Department of Telematics

Problem Description

A context-aware service makes decisions based on the situation (i.e. context) of the involved entities. Context may in general be based on user input (e.g. status in an IM client), sensed (e.g. temperature, location, heart rate), or derived (e.g. combination of location and time of day).

Telenor R&D in Trondheim is involved in several projects where context-aware services are essential. One of these projects is in cooperation with SINTEF on a new system for surveillance and management of fish farms. Development of a context aware service related to this project would consist in the following:

-Gathering physical information (e.g. sea and air temperature, wind speed and direction),
-Persistent storing of this information,
-Distribution of information to e.g. surveillance personnel.

The assignment should contain
-Background study
-Specify, design and implement a context-aware application. Implementation should be based on Java.
-Evaluate the solution

The task may be completed by one or more (up to three) students, and may be carried out as individual assignments for the students involved, or as a joint effort.

Teacher: Per-Oddvar Osland, ITEM
Supervisor: Per-Oddvar Osland and Frode Flægstad, Telenor R&D
Location: The assignment will partially be carried out at the premises of Telenor R&D Trondheim.

Assignment given: 16. January 2006
Supervisor: Per-Oddvar Osland, ITEM

Preface

This thesis report is submitted to the Norwegian University of Technology and Science (NTNU), to fulfil the requirements for the Master of Science degree. The project was carried out at Telenor R&D Tyholt in cooperation with Department of Telematics, NTNU.

The main goal of the project is to create a working prototype of a context based surveillance system for the aquaculture industry. The project is very practical, and that is something we value a lot. It has been a great challenge, but a lot of fun as well. To get the different hardware components working together has not been easy, and some soldering and wiring had to be done. To work with software developed by others have also been quite a challenge, but after a lot of trail and error, and some help, we got the system up and running.

We would like to thank our supervisor Per-Oddvar Osland for tutoring us through this project, and for helpful comments and suggestions. The cooperation has been highly appreciated. We would also like to thank Frode Flægstad and Sune Jakobsson at Telenor R&D for various tips and for supplying the necessary hardware and equipment.

Also Arne Munch-Ellingsen from the University of Tromsø must be mentioned in the list of thanks. He has been responsible for the development of the APMS context manager, and has given us a lot of help during the project. He has also modified and improved the context manager after requests from us.

Trondheim, June 12th 2006

Frank Paaske Jon Arne Grødal

Contents

List of figures

List of tables

List of code snippets

Abbreviations

Table 1: Abbreviations

Abbreviation	Description
API	Application Programming Interface
APMS	(Not yet determined)
CM	Context Manager
CLDC	Connected Limited Device Configuration
CDC	Connected Device Configuration
COA	Connection Oriented Architecture
CoMS	Context Manager System
CORBA	The Common Object Request Broker Architecture
CSV	Comma Separated Values
EDGE	Enhanced Data rates for Global Evolution
EDI	Electronic Data Interexchange
FTK	First To Know
GPS	Global Positioning System
GPRS	General Packet Radio Service
GUI	Graphical User Interface
J2ME	Java 2 Micro Edition
J2SE	Java 2 Standard Edition
J2EE	Java 2 Enterprise Edition
JVM	Java Virtual Machine
MBean	Managed Bean
MDVO	Mobile Dynamic Virtual Organization
ICT	Information and Communication Technology
IDL	Interface Definition Language
IIOP	Internet Inter-Orb Protocol
IM	Instant Messaging

Abbreviation	Description
MTBF	Mean Time Between Failure
MTTR	Mean Time To Repair
MSC	Message Sequence Chart
MIDP	Mobile Information Device Profile
M2M	Machine 2 Machine
NMEA	National Marine Electronics Association
ORB	Object Request Broker
PDA	Personal Digital Assistant
RFID	Radio Frequency Identification
RMI	Remote Method Invocation
RPC	Remote Procedure Call
RUP	Rational Unified Process
SGML	Standard Generalized Markup Language
SDK	Software Development Kit
SIP	Session Initiation Protocol
SMS	Short Message Service
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
TCP	Transport Control Protocol
SCG	Sea Cage Gateway
XML	eXtensible Markup Language
XML-RPC	eXtensible Markup Language – Remote Procedure Call
UML	Unified Modelling Language
UDP	User Datagram Protocol
UTM	Universal Transverse Mercator
VO	Virtual Organization
WTK	Wireless Toolkit

Definitions

Table 2: Definitions

Word or sentence	Description
Sea cage	This is a structure in which fish are held and grown
Net, or closing net	What surrounds and contains the fish in the sea cage
Frame	A collection of sea cages
Fish farm	A collection of frames, operated by the same firm
Sensors	The components that collect information about the environment
APMS Context Manager	A system for collecting, processing, storing and distributing context information Distributes the context from context sources to the context consumers
Input Components	Ensure communication between the context sources and the APMS Context Manager
Output Components	Provides communication between the context consumers and the APMS Context Manager
Context database	Contains all the information about the fish farm, including current and historical sensor information
FiFaMoS	Fish Farm Monitoring System. An acronym describing the whole monitoring system, including context source, context manager and context consumer.
FiFaMoS Context Source	The module that collects sensor information from a sea cage, and feeds it to the context manager.
FiFaMoS Context Consumer	A client application that is used by the operators to monitor the fish farm. The context consumer gets the sensor information from the context manager and displays them in a graphical user interface
FiFaMoS Mobile Context Consumer	A version of the FiFaMoS Context Consumer that runs on a mobile device

Word or sentence	**Description**
M2M module	The module that collects sensor information from a sea cage, and feeds it to the context manager Also referred to as the FiFaMoS Context Source
Web service	A Web Service is a software component that is described via WSDL and is capable of being accessed via standard network protocols such as but not limited to SOAP over HTTP

Abstract

This thesis focuses on context-aware services that make decisions based on the situation (i.e. context) of the involved entities. Context may in general be based on user input, sensed or derived (e.g. combination of multiple context entities). The type of such services is vast, but in this thesis the system is aimed towards the aquaculture industry.

During the last years, aquaculture quality has become more and more important in the fish farming industry. But this importance has not been reflected yet in using information and communication technologies (ICT). The main problem in a fish farm is that most of them are without supervision for a long time while they are exposed to changing weather conditions. This problem gets even bigger when fish farms are established far from land, and often becomes exposed to extreme weather conditions. As a consequence of this, sea cages may work loose, drift away and break. In order to minimize the consequences caused by lack of information on the fish farm (such as weather conditions and other variables) when there is no workers around, ICT surveillance systems should be used. Context-aware services are perfectly suited for this type of application, and the task of this thesis is to specify, design and implement a context-aware application for the aquaculture industry. This includes a context source application, a context consumer application and a service to be deployed on a context management system.

Our solution is named FiFaMoS (Fish Farm Monitoring System) and is based on the APMS context manager. This is a context management system that provides easy service deployment due to built-in support of multiple binding types as well as persistent storing of context. As a context source, an application for an M2M module is developed. There will be one module situated at each sea cage, which collects information like feed level, temperature, pH, oxygen level from the sensors. In addition, the module gets the positioning information for the sea cage from a connected GPS receiver. This information is periodically sent to the context manager that interprets the context and makes it available to users of the system (context consumers). In addition alarms will be triggered if values are out of bounds. It has been developed two different context consumers; one for personal computers and one for mobile phones. In these applications it is possible to view both current and historical sensor values, and receive alarms. It is also possible to alter the fish farm configuration via the PC client.

Aspects that will be discussed are the use of different binding types, representation of data when transmitting and storing, hardware choices and various implementation choices. The implemented FiFaMoS system uses web services as binding to get a loosely coupled system, and objects are represented in XML which makes the system easy to alter. Detailed testing has been performed, and the system works as intended.

I really rather care for fish
In fact they are my favourite dish

I like to take small bits of dace
And put them right into my face

My midday snack's a stickleback
My evening meal's a conger eel

If I were given just one wish
I'd have the world knee deep in fish

An excerpt from “Number 101”
by Graham Chapman and John Cleese

1 Introduction

1.1 Motivation

The fish farming industry has grown to become an important Norwegian export industry, and one of the major employment sectors along the Norwegian coast. A single fish farm may hold values for tens of millions NOK, and incidents such as disease, accidents, improper breeding conditions etc may lead to heavy expenses and environmental consequences. Occurrence of such incidents may be reduced by using frequent surveillance, this is however labour intensive and therefore costly. One solution could be to apply remote surveillance by making use of sensors, positioning systems and cameras.

Several trends in the industry call for remote surveillance:

- Because of environmental issues, fish farms will have to be moved from in-fjord locations to costal locations. This implies relocation from safe spots like inner fjords and calm coast areas to areas that are more exposed to extreme weather. And because workers only are on-site for 2-4 hours a day, an accident will not be discovered for up to 20 hours. And in some instances it may take days between each inspection, which makes the situation even worse.

- Small fish breeding actors are being merged with or bought by larger actors, resulting in fewer and larger fish farm installations. More values are collocated, and the expenses put in surveillance equipment will pay off more easily.
- High labour costs, long and dangerous travel distance to fish farms in costal/offshore locations

A system where information is collected and distributed separately and selectively is the optimal solution. This way the user (or whoever needs to see the information) can simply select which information to see, not needing to search for it.

1.2 Scenarios

In these scenarios, possible cases that can happen in a fish farm will be analyzed. Two scenarios will be described, each with two different outcomes. One based on today's situation (scenarios 1A and 2A) and one after a fish farm monitoring system is installed (scenarios 1B and 2B). The scenarios are based on the fish farm in Figure 1, where the surveillance system only will be available in the tomorrow section of the scenarios.

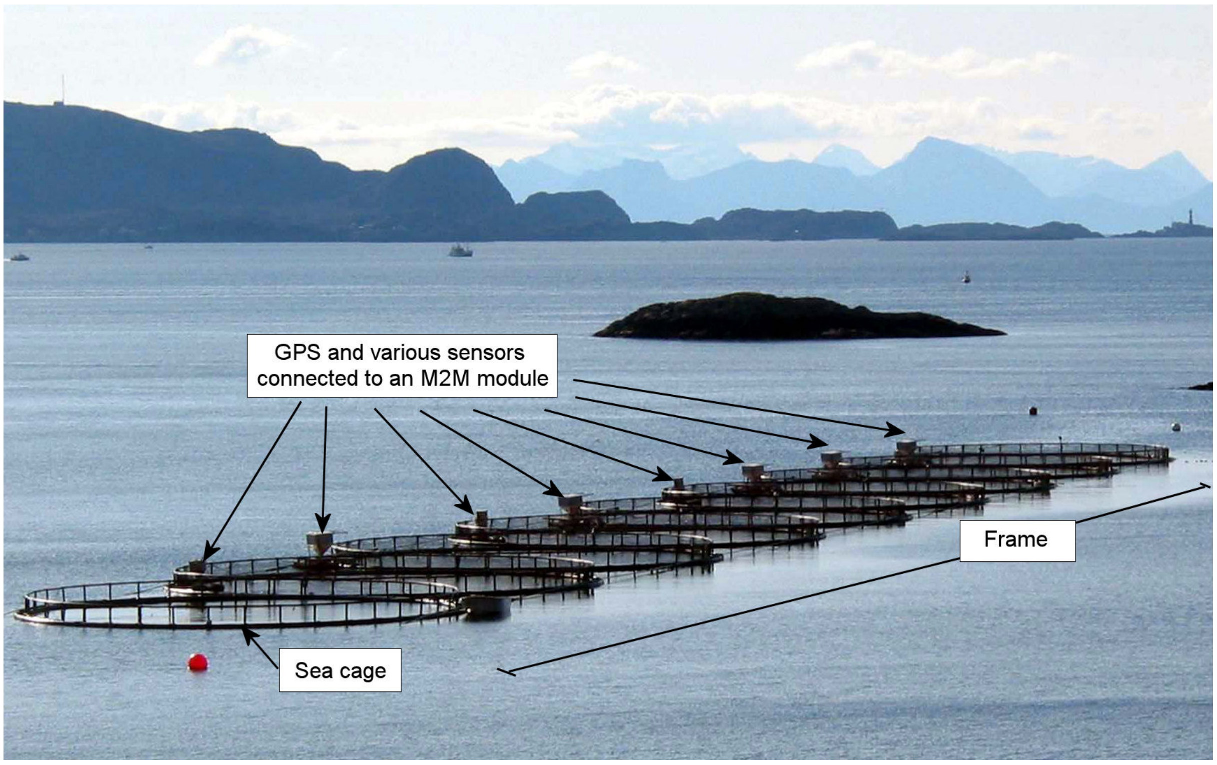


Figure 1: A fish frame with sea cages and sensors

Figure 1 shows a fish frame that contains nine sea cages. A fish farm can have several frames that each has various numbers of sea cages. In this project, there will be one sensor controller per sea cage. The controller will be an M2M GPRS module, with a GPS connected. Various sensor types can be connected to this module that can pass the sensor information via the Internet, and made accessible to clients. These clients can monitor the sensor values of the fish farm.

Each sea cage has many attributes that need to be monitored. Table 3 shows some of the sensor types that can be useful on a fish farm, and the subchapters describe scenarios concerning position and food level.

Table 3: Sensor types that can be useful in the aquaculture industry

Sensor type	Purpose
Position (GPS)	Alarm the operators if a sea cage is loose and drifting away
Sea cage net sensor	Find out if the sea cage net is dirty due to growth. This stops clean water from flowing through the sea cage. The sensor detects if cleaning is necessary.
Water current sensor	Measures the tidal currents in the fish farm. This information can be used to prevent feeding during excessive currents, thus preventing feed waste.
Water quality sensors	Measure various water quality parameters
Temperature sensor	Measure the water and/or air temperature
Food level	Measure the food level in the feeding tanks
Pellet sensor	Finds the amount of food that is not eaten. Feeding routines can then be adjusted.
Camera	Video of the sea cage can be used to detect uneaten food, and to see if the fish is behaving as usual.
Wave sensor	Measure the height of the waves
Water level sensor	Measure the depth of the water
Wind sensor	Measure wind strength and/or direction

1.2.1 Scenario 1-A: Today - A sea cage drifts away

Alice, the environmentalist, has been active in the process of moving fish farms from calm fjords to rougher sea. One stormy day in January, one of the sea cages in Bob's fish farm works loose, and starts drifting towards sea. The sea cage is a new high-tech steel cage containing almost 200 tons of fish. The next day when Bob takes his visit to the fish farm, he discovers that the sea cage is gone, and immediately starts a search with his boat. However, the sea cage has drifted a long way, and Bob is unable to find it. Bob has lost almost 200 tons of fish, and Alice is angry because the sea cage pollutes the environment.

1.2.2 Scenario 1-B: Tomorrow - A sea cage drifts away

The situation is the same as in scenario 1-A, but now, Bob has invested in a fish farm monitoring system with a GPS sensors on each sea cage.

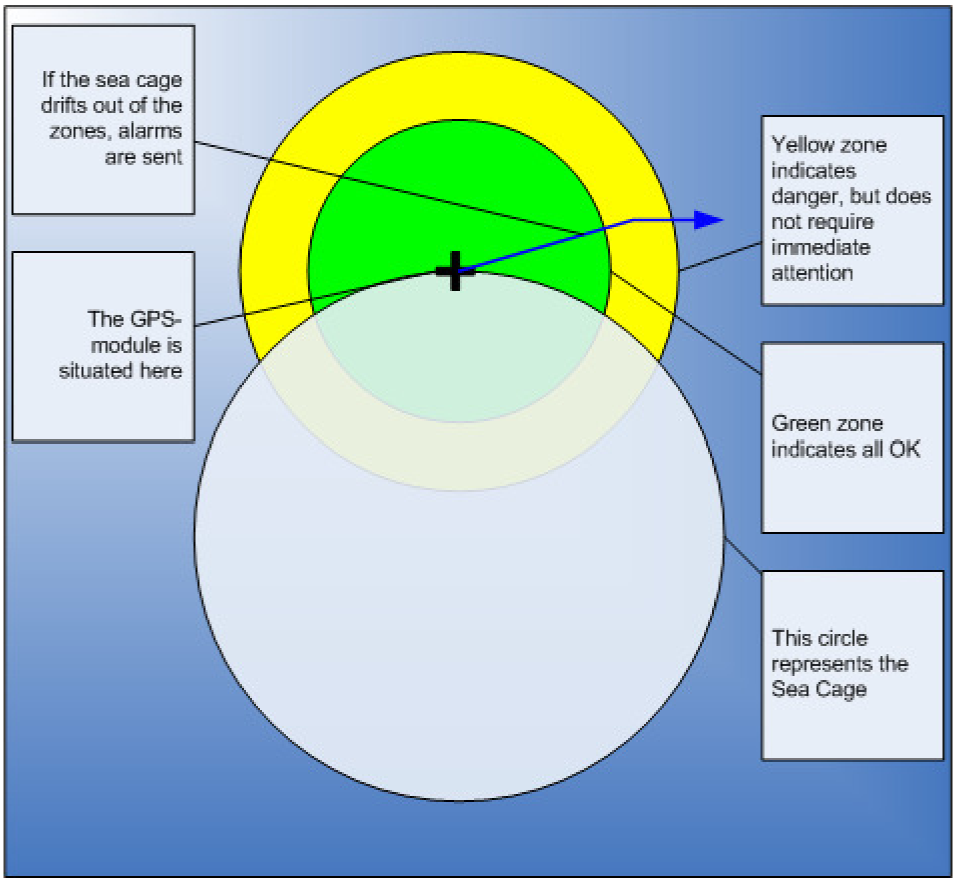

If the sea cage drifts out of the zones, alarms are sent
The GPS-module is situated here
Yellow zone indicates danger, but does not require immediate attention
Green zone indicates all OK
This circle represents the Sea Cage

Figure 2: A sea cage in its right place, but with a blue arrow indicating a drift towards the yellow zone.

With a GPS sensor and a M2M GPRS module, the position of the sea cage will always be well known. Immediately after the sea cage works loose, Bob gets an SMS alarm on his mobile that says that the sea cage is out of position. After checking the positioning information on the map, Bob hires a towing boat, and finds the sea cage. Luckily, the net is intact, and the sea cage can be set in place again. 200 tons of fish is saved, and the environment is preserved.

1.2.3 Scenario 2-A: Today - Food level in feed tank

Just after Bob's daily visit to the fish farm, the feed tank on one of the sea cages bursts open, and the feed sinks to the bottom of the sea. The fish have nothing to eat all day, and optimal growth is prevented. Some of the fish even get sick, and die. The disaster is not revealed before Bob's next daily visit, so it has been 24 hours without feed. In addition Alice is not happy, because the feed pollutes the environment around the fish farm.

1.2.4 Scenario 2-B: Tomorrow - Food level in feed tank

Bob has invested in a fish farm monitoring system with sensors measuring the food level in the feed tanks. When the feed tank bursts open, the feed level sinks as shown in Figure 3.

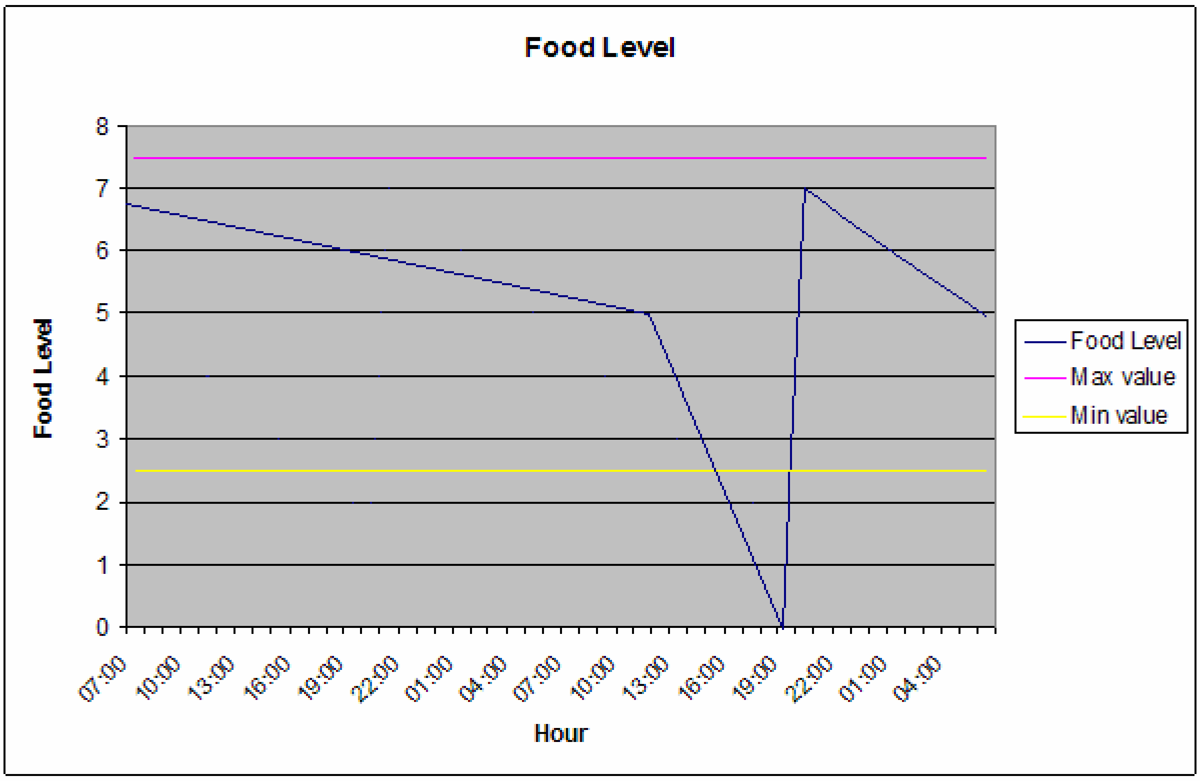


Figure 3: The food level for a sea cage. The graph shows the level of food in a feeding tank for a sea cage.

At 16:00, the food level has exceeded the minimum value, and with the help of the fish farm monitoring system, Bob will get an alarm. He takes a trip to the fish farm with his boat, and reveals the damage. The old feed tank is immediately replaced with a new one, and the fish are happy because they still have food.

If special feeding sensors and controllers are present, it is also possible to adjust feeding on the base of estimation of fish feeding activity. Monitoring the food level in fish farms can increase fish growth efficiency and enhance the ecology around the place of the fish farm. As a result, it decreases food expenses and environmental pollution by giving the fish the amount of feed needed for optimal feed assimilation.

1.3 Problem statement

A context-aware service makes decisions based on the situation (i.e. context) of the involved entities. Context may in general be based on user input (e.g. status in an IM client), sensed (e.g. temperature, location, heart rate), or derived (e.g. combination of location and time of day).

Telenor Research and Development are currently working on a new system for surveillance and management of fish farms [4] [5]. This project is aimed towards the Akogrimo mobile grid project [1], and is being realised by two master students. A new context management system called APMS [6] is implemented by a team at the University of Tromsø, and further knowledge about this system is wanted. By utilizing this system, an alternative solution for fish farm monitoring shall be introduced. The system will make the values from the sensors described in section 1.2 available to various clients. These clients shall present the values in a user friendly way. Sensor values shall be stored in a database, so that historical sensor values can be presented.

A context-aware service for surveillance of fish farms shall be designed, implemented and demonstrated. Implementation shall be based on Java, and the service shall utilize the APMS middleware system.

1.4 Limitations of the thesis

Due to the limited time scope of the project, some limitations will have to be made. These are described in this chapter.

The system implemented shall only be a prototype, and not be tested in a realistic off-shore environment. The sensors values used when testing shall not be produced by real fish farm sensors, but by sensor simulators or dummy sensors (for instance variable resistors). This is to limit the scope of the thesis, and concentrate mostly on the communication part of the project.

Security issues are also given a low priority. The main goal of the thesis is to create a working system that demonstrates context-aware services, and AAA functionality is not needed.

Even though financial aspects are an important matter in a real-life project, it has not been given much attendance. These aspects are considered to be outside the scope of this project.

1.5 Related work

Within the scope of the EU project Akogrimo [1] [3], an ICT-system for remote monitoring and controlling of fish farms is being developed. The project is called Sea Cage Gateway, and is being realized by different diploma thesises given by Telenor Research and Development. The goals are very similar to our project, but the solution is based on a different context management system using mobile grid technology.

The work covers the following key elements:

- Identify actors and roles in the fish farming value network that would profit from building a virtual organization (VO).
- Characterize relevant business activities within the mobile dynamic virtual organization (MDVO) with a special focus on commercial exploitation of mobile grid inventions, such as accounting capabilities or context management.
- Develop a demonstrator that visualizes sea cage gateway with its actors and business flows.

Another aspect of the sea cage gateway project is the sensor networks. The project shall analyze the needs for sensor data to be collected, checked, and passed on to a control station. This is a node where the main data analysis and business logic is done.

Tendo Tech AS is a company developing surveillance systems for mobile devices. Their system is very flexible, and is used in many scenarios, including fish surveillance. This version of the system, FTK Marin, does many of the same things this project will do, but is made for indoor fish farming. Therefore it is not possible to get positioning information for sea cages, and the sensor network installation is not customized to handle harsh conditions off-shore. Another important difference is that the FTK Marin system is based on an instant messaging system called Jabber [31]. This makes the fish farm controllable from a special instant messaging client on the mobile phone. A drawback with this solution is that it is not possible to store sensor data persistent, so historical sensor values can not be viewed.

AKVAsmart ASA is a technology company with activities in the fish farming industry. It is the world's leading supplier to the salmon farming industry within both of its areas: "Farm Process Technology" and "IT & Consulting". In addition to feeding technology, the company also delivers production management systems, and the latest addition is the FishTalk project. The main purpose of FishTalk is to be an integrated solution where all information is stored at site-level. This database information includes feeding and environmental sensors in addition to other information that affect decisions in the fish farm. FishTalk functions include:

- High-level reporting and analysis
- Site Control
- Feeding Process Control
- Enterprise planning
- Maintenance
- Quality Assurance
- Tracing
- Budgeting and Cost Analysis

1.6 Report organization

This report is organized into five main parts: Introduction, theory, realization, discussion and conclusion. The content of the different parts is as follows:

Section 1, Introduction, will introduce the fish farm industry, their needs and describe the problem statement. Section 2, Theory, will give background information on context and context-aware services, and in addition describe the most important technologies used during the project. This includes the APMS context manager, M2M technology, programming environments, communication protocols and positioning technology. Next, section 3, Realization of FiFaMoS, describes the entire development process. This includes different phases like: Prestudy, requirements specification, design, implementation and testing. During the project, several decisions had to be made. That included both software and hardware options in addition to the architectural and implementation choices. Section 4, Discussion, will describe these choices. In the end, section 5 concludes the thesis report.

2 Theory

This chapter describes the background theory of the project. First various aspects of context are described, then the related technologies of the project are examined. Related technologies include the APMS context manager, Java, XML, web services, GPRS, the global positioning system, and sensor technology.

2.1 Context

This section describes what context and context-aware services are. The main focus will be on surveillance services and sensed context. Finally, a three-layer architecture for context-aware services will be introduced and described.

2.1.1 Defining context

The Word Reference Dictionary defines context as:

"*The set of facts or circumstances that surround a situation or event; the historical context*" [8].

However, context is a wide notion with different meaning in different fields. A definition that is clearer in our field of interest is proposed by Dey:

"Context is any information that can be used to characterize the situation of an entity. An entity is a person, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves." [9].

This information, or as we now say, context can be of several types [10]:

- User context: role, identity, location, preferences, social situation, permission profile
- Computing context: network connectivity and nearby resources (printers, displays, and workstations)
- Time context: time of a day, week, month, and seasons of the year
- Physical context: such as weather and temperature
- Context history: when the user, computing, and physical contexts are recorded across a time span, we obtain a context history

In addition a derived context type can be added to the list. This is context information that can be computed on the fly. Obvious examples might be alarm information or network activity.

One can say that context can be divided into two main types: User set context and externally set context, where externally set context includes computing context, time context, physical context other context information not set by the user. Typical surveillance systems like the one discussed in this thesis mostly deal with externally set context from sensors (sensed context).

2.1.2 Characteristics of sensed context

In [11], Henricksen et al. describe some general characteristics about context information in pervasive environments. This section transfers some of these characteristics into the surveillance category, and describes sensed context.

2.1.2.1 Sensed context information is imperfect

Surveillance environments are highly dynamic, which means that information describing them can quickly become out of date. This is both because the context needs processing, and because sources, repositories and consumers of context are distributed over networks. These factors can lead to large delays between the production and use of context information, and lead to that wrong context is used.

Another example that shows use of wrong context is when sensors or derivation algorithms provide faulty information. Sensors cannot represent reality in a perfect way, and derivations/conversions often expand the level of error. This is particularly a problem with context information from crude sensor inputs; like two-level digital sensors and RFID positioning information.

It is also a possibility that sensed context is wrong because of malfunctioning sensors or cut in the path between the context source and the context consumer. This will make parts of the context, or all of the context unknown.

2.1.2.2 Alternative representations

There is usually a significant gap between sensor output and the level of information that is useful to applications, and this gap must be bridged by various kinds of processing of context information. For example, the GPS sensor in scenario 1-B may supply raw longitude and latitude coordinates, whereas an application might only be interested in if the sea cage is in position or not. Using the UTM system for representing the position is also an alternative.

2.1.2.3 Context stack

Just as computer networks can be viewed as a seven-layer model according to the Open System Interconnect (OSI), context can be represented in a four-layer model. This model is proposed by Li [13], and promotes the understanding of context information acquisition.

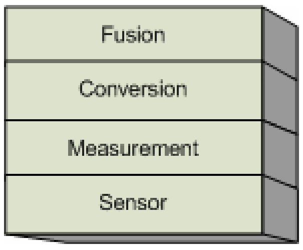


Figure 4: Context stack

The four layers have the following tasks:

- Sensor layer: This is the physical layer where the sensors and the corresponding drivers are located. The sensor layer offers access to raw context data (for instance the temperature)
- Measurements layer: This layer has algorithms to take raw sensor data into more useful data types (for instance a voltage level or digital frequency representing the temperature).
- Conversion layer: This layer converts the raw data into data types more suited for the domain of origin (for instance the number of degrees Celsius).
- Fusion layer: The most abstract layer in the model. The data is grouped into different categories, and might be harvested from several different sensors (for instance the number of degrees Celsius combined with the time the temperature was sampled).

2.1.3 Context-aware services

Context-awareness is a term that is used for devices or systems that have information about their context, and can react accordingly. Context-aware devices may also try to make assumptions about the user's current situation. An example is a context-aware mobile phone that knows that it is located in a meeting room. The phone may conclude that the user is currently in a meeting and reject any unimportant calls. This example goes under the pervasive/ubiquitous computing category, but also other types of context-aware services exist.

An important variant of a context-aware service is surveillance systems like the one treated in this thesis. In these systems, clients can access the sensed context, and be warned when changes occur. This project is about making a surveillance system for the aquaculture industry, and context-aware services are ideal for distributing sensor information to several clients.

The context information in the system is mainly sensor information from the different sea cages. This is externally set context that can be placed in the awareness module layer in the three layer architecture in Figure 5.

2.1.3.1 Three Layer Architecture

Indulska et al. [12] suggest a three layer architecture that consists of a set of interacting components (or modules) organised into three layers. This is shown in Figure 5. In the proposed system, the different layers communicate via a message based notification service called Elvin [15]. This is a client-server architecture that enables distribution transparency, and lets the modules be placed on different locations. However this kind of three layer architecture can be valid for many context-aware systems, including the one used in this project, and the Elvin service is not necessary to use to get the distribution transparency. Using for instance SIP or Web-service messages will accomplish the same.

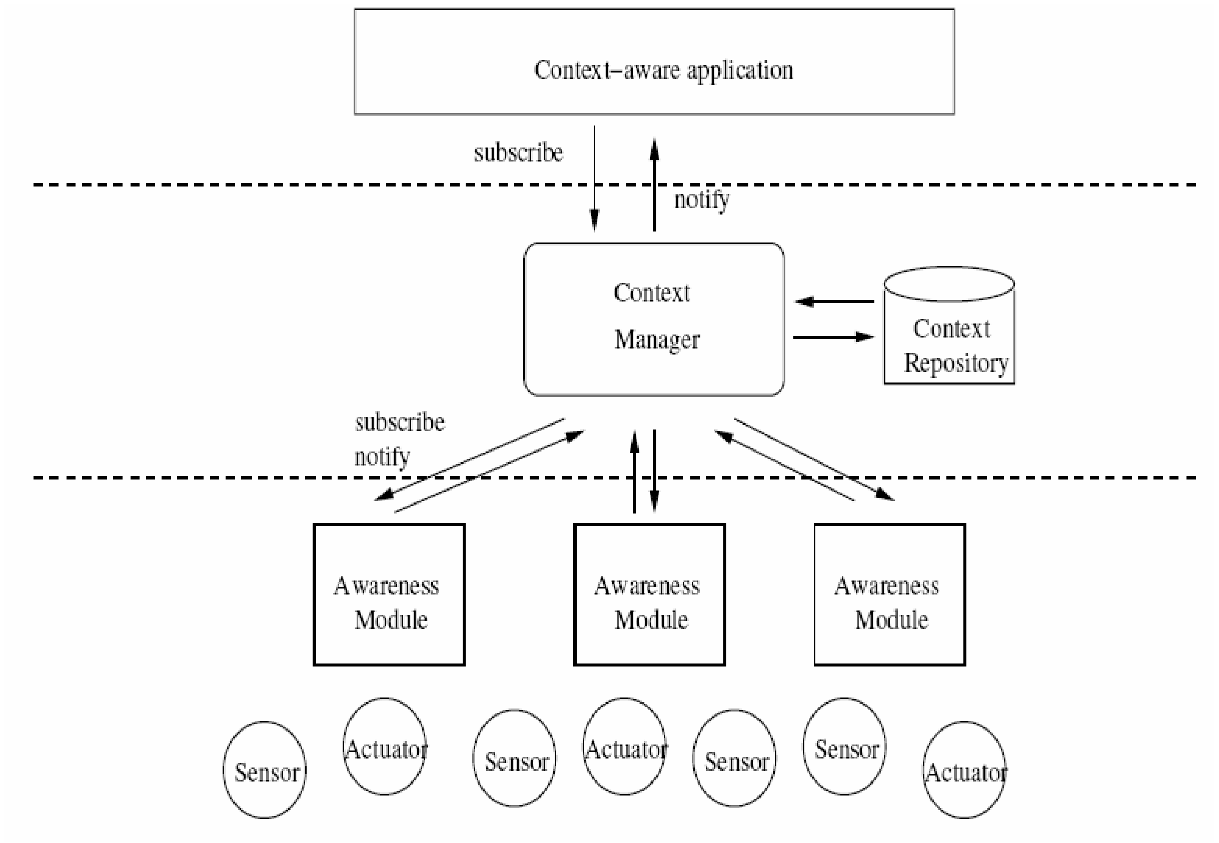


Figure 5: Context management architecture [12]

The main purpose of the different layers is as follows:

Context-aware Application layer: In order to be aware of the current context, the adaptation engine of the pervasive system and/or the context-aware application can subscribe to a set of events of interest (e.g. context changes). These clients are then notified by the Context

Manager upon occurrence of the events of interest. In some context-aware systems, the clients aren't notified about new context information, but have to continually ask for the context information wanted. This is a matter of technological choices.

Context Manager Layer: The Context Manager manages the persistent repository (database) of context information, receives context updates from the Awareness Module layer, and makes the context available to the context-aware applications.

Awareness Module layer: These modules process context information from sensors. They monitor sensors and actuators. An awareness module is a specialised module that provides a particular functionality. For example, it can serve a GPS sensor that provides location coordinates or an actuator that locks a door. Awareness modules transform information from sensors into a format that can be understood by the context manager. The communication between Sensors/Actuators and the awareness modules are often proprietary (e.g. a specific protocol that uses RS-232).

2.2 Enabling and related technologies

Here, the most important technologies used in the system will be described. This includes the middleware system, the programming languages, data representation, positioning and network technologies.

2.2.1 APMS – The context management system

Without a well defined programming model for context-aware service development, it is a tedious task to create even the simplest context-aware application. Tasks such as binding to context sensors, storage of ingested sensor data, context information inference etc. should be supported by the underlying infrastructure. Context management middleware makes it easier to include context-awareness in applications since support for bindings, persistence and inference is part of the container support ordered to container components.

The context manager that will be used in our system is called APMS. According to [6], it is a container based middleware system, similar to JBoss. It is used to support the basic needs of a context enabled system. Examples of such needs are:

- Collection and distribution of context (see 2.2.1.1)
- Persistence of context information (see 2.2.1.2)
- Integrated generation of dynamic content (see 2.2.1.3)
- Automatic trigging of actions (see 2.2.1.4)
- Rapid deployment and easy configuration

APMS is based on managed beans, or MBeans, which acts as wrappers for applications, components or resources in a distributed network. Therefore APMS also supports the usage of Java Management Extensions (JMX) [7]. This allows a centralized management of the MBeans. This functionality is provided by an MBean server, which serves as a registry for all MBeans, exposing interfaces for manipulating them. In addition, JMX contains a service that allows dynamic loading of MBeans over the network. In the JMX architectural model, the MBean server becomes the spine where all the server components plug in and discover other MBeans.

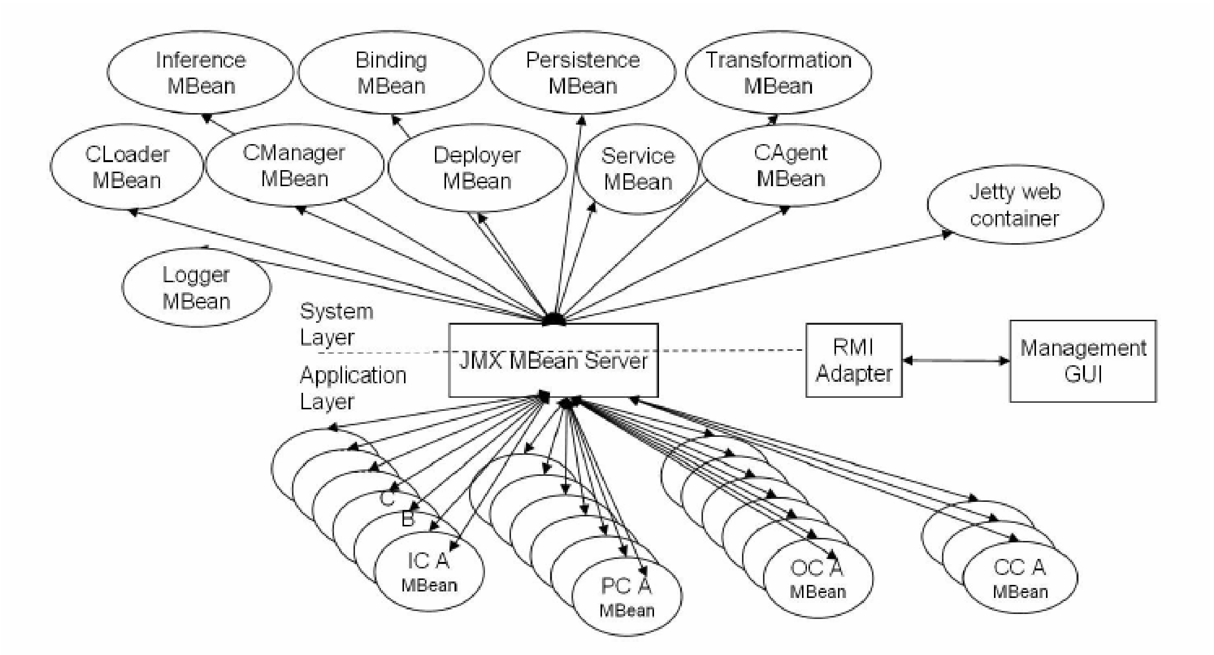


Figure 6: APMS container architecture

2.2.1.1 Components

In this section we will show what the different components are used for. Most components are self-explanatory, but it's still necessary to clarify the usage to make sure that no mistakes are made. A component is made up by an interface named *MBean.java, a meta-file (XML) named *Meta.xml and an implementation of the MBean-interface named *.java. The asterisk (*) represents the name of the component. The name is often used to indicate what kind of component it is. An example could be InputMBean.java, InputMeta.xml and Input.java.

2.2.1.1.1 Input component

The input component is used for external entities to provide data *to* the service. Typical usage would be as an interface for sensors to send their values to the database. This component may use one of the following methods to make it connectable:

- Web Services (XML-RPC)
- Sockets (UDP/TCP)
- Remote Method Invocation (RMI)

The chosen method is specified in the component's meta-file by the programmer, and must be used accordingly.

2.2.1.1.2 Process component

The process component is used to refine and increase value of input data. This could be converting values from mV to wind speed (m/s), degrees centigrade (°C) or acidity (pH). The process component defines a set of rules that will trigger a predefined action if the rule checks out (see 2.2.1.4 for more information on the rules engine).

2.2.1.1.3 Output component

The output component is used for external entities to retrieve data from the service. Typical usage would be as an interface for applications to get stored sensor values from the database. To make the it connectable, the output component may use the same methods as the input component.

2.2.1.1.4 Configuration component

The configuration component is used for configuring external entities. Typical usage would include setting refresh value, host, port and timeout for a connection. By using a configuration component the configuration can be done via the web interface (see appendix C.2).

2.2.1.2 Apache Derby database

Apache Derby [30] is a database completely implemented in Java. It is based on the SQL/JDBC standards and support most database features including tables, indexes, views, triggers, sub-queries, procedures, functions, transactions, isolation levels, encryption, etc. The database is used in the APMS context manager to store context information. In the APMS system it is possible to describe the tables and relations with XML in the meta-files of the components.

2.2.1.3 Jetty web server

Jetty [29] is a java-based stand-alone HTTP server and servlet container. This means that you do not need to configure and run a separate web server in order to use java, servlets and JSPs to generate dynamic content. Jetty is embedded in applications and products without adopting the WWW centric application architecture. This also involves the APMS middleware system that uses Jetty as the receiver and sender of web service requests and responses. In addition it is used to host web sites and web interfaces associated with the service. Other applications servers that utilize Jetty are Geronimo, JBoss and JOnAS.

The web server and web application run in the same process, without interconnection overheads and complications. Furthermore, as a pure java component (configured in a *jar* file under 350KB), Jetty can be simply included in the application for demonstration, distribution or deployment.

2.2.1.4 Drools

The usage of a rules engine is a very efficient way to collect decision-making logic and work with large data sets. A rules engine can make decisions based on thousands of facts much more quickly and reliably than humans, and is therefore the preferred choice for decision making.

Drools [32] is a rules engine implementation based on Charles Forgy's *Rete* algorithm adapted to Java. Adapting *Rete* to an object-oriented interface allows for more natural expression of business rules with regards to business objects. Drools is written in Java, but able to run on both Java and .Net.

Drools is flexible enough to match the semantics of a problem domain with Domain Specific Languages (DSL) via XML using a defined Schema. A DSL consists of XML elements and attributes that represent the problem domain, as shown in Code 1.

```
<rule name="Generate alarm">
    <parameter identifier="sensorValue">
        <class>SensorValue</class>
    </parameter>
    <parameter identifier="sensor">
        <class>Sensor</class>
    </parameter>
    <java:condition>
         sensorValue.getValue() &gt; sensor.getMaxValue()
    </java:condition>
    <java:consequence>
      sensor.generateAlarm(sensorValue);
    </java:consequence>
</rule>
```

Code 1: Example of a rule defined in a DSL

2.2.2 M2M - Sensor information acquisition

To acquire context/sensor information, various solutions could be used. Among them are custom built equipment, industrial computers and M2M technology. Since the latter technology offers high flexibility at a low price and size, M2M modules are well suited for surveillance tasks.

A M2M module is basically a mobile phone with no display or keyboard, but a number of digital and analogue inputs and outputs. These inputs and outputs together with the GPRS Internet connection make it suitable for transferring context information over the air from almost any location.

A number of different M2M modules are on the market. Most of them have got serial connection ports, and can be connected to a GPS for positioning purposes. However it is more convenient to have the M2M module bundled with the GPS in one single package. The serial port can also be used to connect any other serial based peripheral equipment, including cameras, weather stations and household power controllers. When it comes to attaching sensor, the number of inputs is an important parameter. Most of today's modules have only a handful of analogue inputs, this can often be a bottleneck. However, if a relay module is used, several sensors could be connected to one input. The relay could then be controlled by the digital outputs on the module. This way the module can choose which sensor it wants to read. Connecting several sensors to a hardware module that converts the sensor information to a proprietary RS-232 format before sending it to the M2M module via the serial port is also a possibility.

2.2.2.1 Sensors for M2M modules

Most M2M modules have a couple of analogue inputs and digital inputs. The analogue inputs are set by a voltage in the area 0-2800mV. Not many sensors output this kind of voltage, and we often have to use special circuits that can transform the sensor output to the voltage level wanted by the M2M module.

The National Semiconductor L45 is an example of an integrated circuit that outputs a voltage linearly proportionally to the temperature.

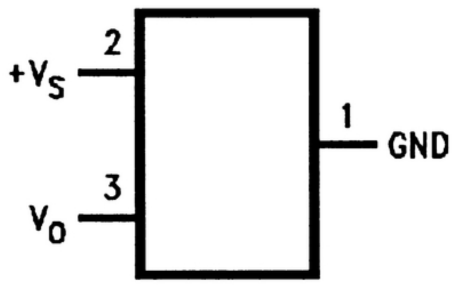


Figure 7: L45 temperature sensor integrated circuit

When feeding a +5V voltage to the V_S pin, the V_0 will give a voltage between 0V and 3V, depending on the temperature (that can be from -20 °C to +100 °C). This will work well for measuring in-house temperatures, but rough surroundings as the ocean would require a more designated sensor.

The SmarTec SMT160-temperature probe is more roughly packaged, and can stand to be stored under water. However this probe does not give an output voltage proportional to the temperature, but a digital frequency that varies.

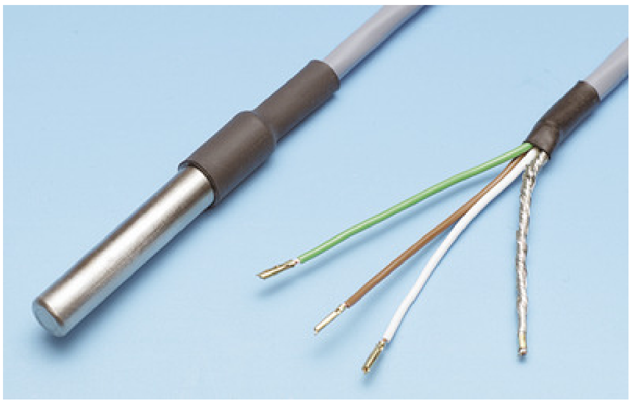


Figure 8: SmarTec SMT 160 temperature probe

This way of outputting a digital pulse with frequency that varies according to the sensed value is very normal in industrial solutions, and M2M solutions would often require hardware between the sensor and the module that either interprets the digital pulse to an analogue voltage, or to serial data that the module can read.

AKVAsmart (mentioned in section 1.5) also make sensors for the aquaculture industry. These sensors could probably be used in an M2M system, with the help of some hardware integration.

Table 4: AKVAsmart sensor types [14]

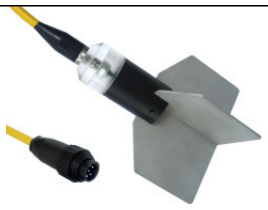
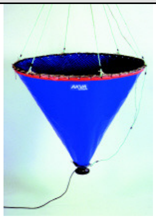
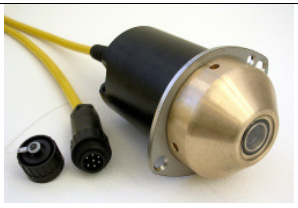
Figure	Sensor description
	The oxygen sensor measure the level of oxygen contained in the water. This parameter is one of the most important when it comes to fish welfare.
	The water current sensor can measure the current strength and direction. This information can be used to stop the feeding when the current it strong, thus preventing feed waste.

Figure	Sensor description
	The feed waste sensor measures the amount of feed not eaten, and the feeding can be adjusted to save feed and to save the environment.
	By using a underwater camera, the condition of the fish and feeding response can be monitored.

2.2.3 Java

This project will be implemented using Java Technology. Applications for computers will utilize Java Standard Edition (Java SE, formerly known as J2SE), applications for mobile devices will use Java Micro Edition (Java ME, formerly known as J2ME)

2.2.3.1 Java Standard Edition

Java Standard Edition offers a complete environment for application development and deployment on desktops and servers. Java SE is also used in today's embedded and real-time environments. The Java programming language is syntactically similar to C++ but differs fundamentally. While C++ uses unsafe pointers and programmers are responsible for allocating and freeing memory, the Java programming language uses type safe object references, and unused memory is reclaimed automatically. Furthermore, the Java programming language avoids multiple inheritance to get cleaner interfaces.

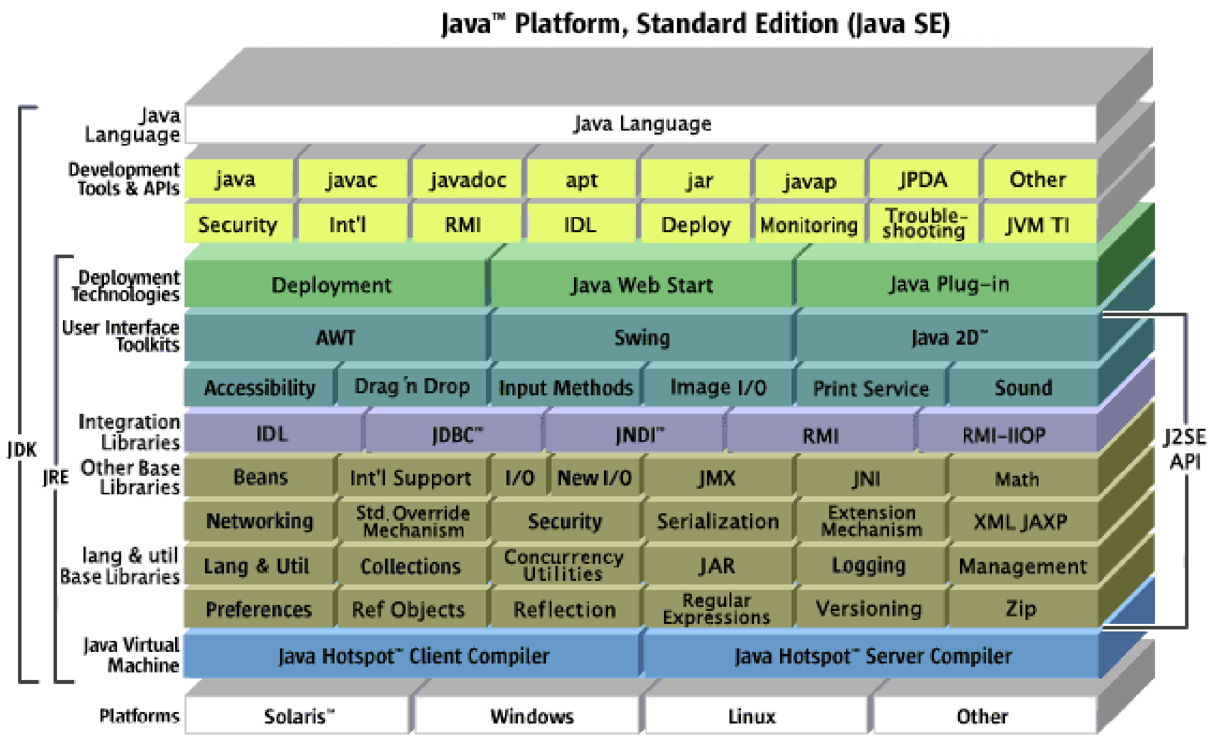


Figure 9: Java Standard Edition block diagram

Figure 9 illustrates all the component technologies in Java SE platform and how they fit together.

2.2.3.2 Java Micro Edition

Java Micro Edition provides a robust, flexible environment for applications running on consumer devices, such as mobile phones, PDAs, TV set-top boxes, printers, M2M modules and a broad range of other embedded devices. Like its counterparts for the enterprise (Java EE) and desktop (Java SE), Java ME includes Java virtual machines and a set of standard Java APIs.

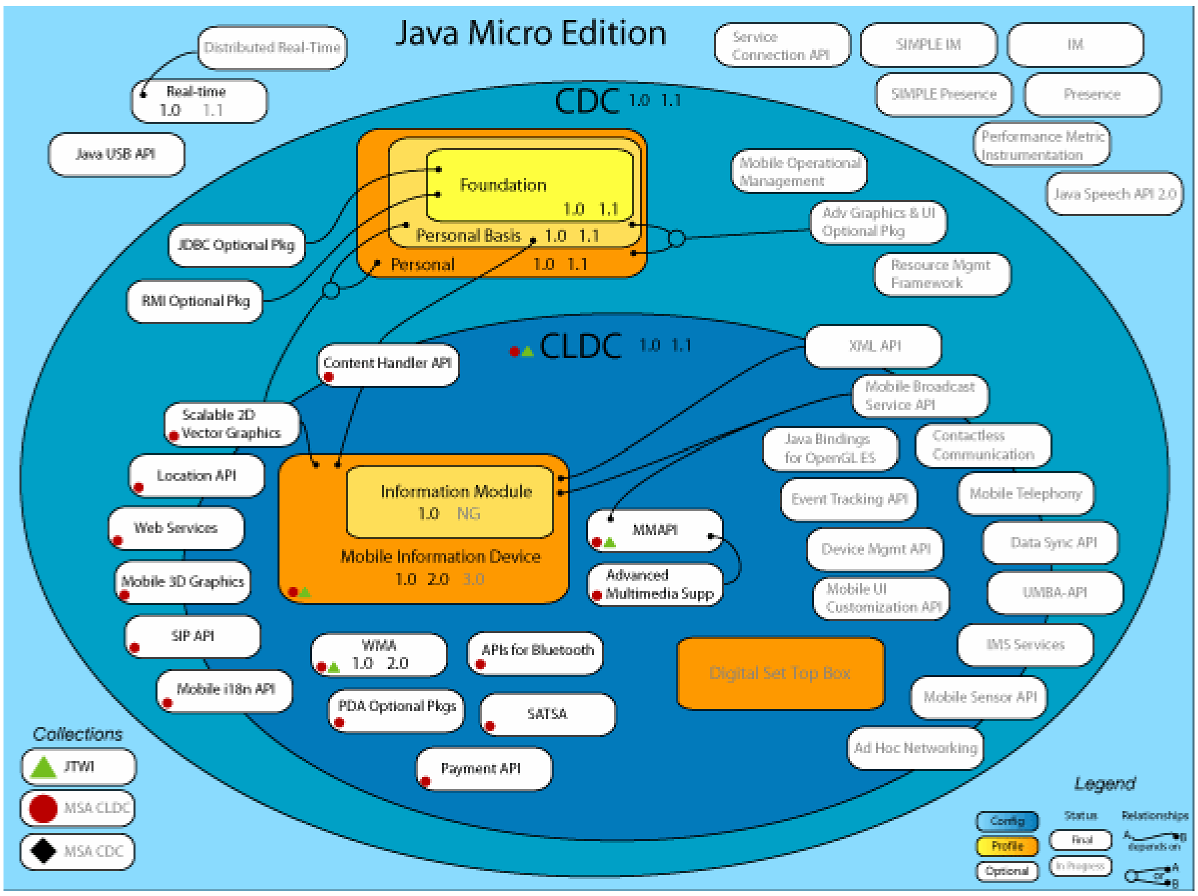


Figure 10: Java Micro Edition overview

As shown in Figure 10, Java ME has two base configurations. The first is the Connected, Limited Device Configuration (CLDC). This configuration is for small wireless devices with intermittent network connections, like mobile phones, and personal digital assistants (PDAs). The Mobile Information Device Profile (MIDP), which is based on CLDC, was the first finished profile and thus the first finished Java ME application environment. MIDP-compliant devices are widely available worldwide.

The other base configuration is the Connected Device Configuration (CDC). This configuration is for larger devices (in terms of memory and processing power) with robust network connections. Set-top boxes, Internet appliances, and embedded servers are good examples of CDC devices, although high-end mobile devices also fit this configuration well. The Foundation Profile extends CDC and serves as the basis for several other profiles. It provides core APIs shared with Java SE, including classes and interfaces from *java.lang, java.io, java.security, java.util*, and more.

2.2.4 eXtensible Markup Language (XML)

From the specification [42], XML can be defined as the following:

"The Extensible Markup Language (XML) is a subset of SGML (Standard Generalized Markup Language (SGML) that is completely described in this document. Its goal is to enable generic SGML to be served, received, and processed on the Web in the way that is now possible with HTML. XML has been designed for ease of implementation and for interoperability with both SGML and HTML."

More generally, XML can be said to be a way of describing data and an XML file can contain the data too, as in a database. Its primary purpose is to facilitate the sharing of data across different systems, particularly systems connected via the Internet.

2.2.5 Web services: XML-RPC

According to the World Wide Web consortium, web services are defined as the following:

"A web service is a software application identified by a URI, whose interface and bindings are capable of being identified, described and discovered by XML artefacts and supports direct interactions with other software application using XML based messages via Internet-based protocols"

In other words, it is a language and platform independent method to implement Service Oriented Architecture (SOA) using standard Internet technologies. Web services are used for application-to-application communication, and several technologies exist. The one used in this project is XML-RPC.

XML-RPC is short for XML Remote Procedure Call, and is a protocol for performing remote procedure calls over HTTP. XML-RPC makes it possible for applications to communicate and share data over the Internet, no matter what platform, hardware or programming language used. The protocol can use HTTP port 80, so it can traverse most firewalls in the Internet. The protocol has three basic message types: The request type, the response type and the error type.

It is a simple protocol, and supports only a number of data types. The available types are described in Table 5.

Table 5: XML-RPC data types

Data type	Tag	Description/example
Integer	`<int>6</int>` or `<i4>6</i4>`	Whole number
Double	`<double>2.3</double>`	Floating number
String	`<string>Text</string>`	Text string
Boolean	`<Boolean>0</Boolean>`	Logical value, 0 or 1.
Date/time	`<dateTime.iso8601>2004-05-24T09:30:59</dateTime.iso8601>`	Date and time value
Base64	`<base64>eW91IHJlYWQgdGhpcyE=</base64>`	Base 64 encoded data
Array	`<array>` `<data>` <- mandatory child of array element `<int>123</int>` `<double>-12.345</double>` `<array>` `<data>` `<dateTime.iso8601>20040524T09:30:59<dateTime.iso8601>` `<string>hello soap</string>` `</data>` `</array>` `</data>` `</array>`	Array of values, storing no keys
Struct	`<struct>` `<member>` `<name>paper code</name>` `<value>0657312B</value>` `</member>` `<member>` `<name>paper title</name>` `<value>Something or Other</value>` `</member>` `</struct>`	Array of values, storing keys
Null	`<nil/>`	Discriminating null value

2.2.6 CORBA

The Common Object Request Broker Architecture (CORBA) [48] is a distributed object computing infrastructure being standardized by the Object Management Group (OMG).

CORBA automates common network programming tasks such as object registration, location, and activation; request demultiplexing; framing and error-handling; parameter marshalling and demarshalling; and operation dispatching. Using the standard Internet Inter-Orb Protocol (IIOP), a CORBA-based program from any vendor, on almost any computer, operating system, programming language, and network, can interoperate with a CORBA-based program from the same or another vendor, on almost any other computer, operating system, programming language, and network.

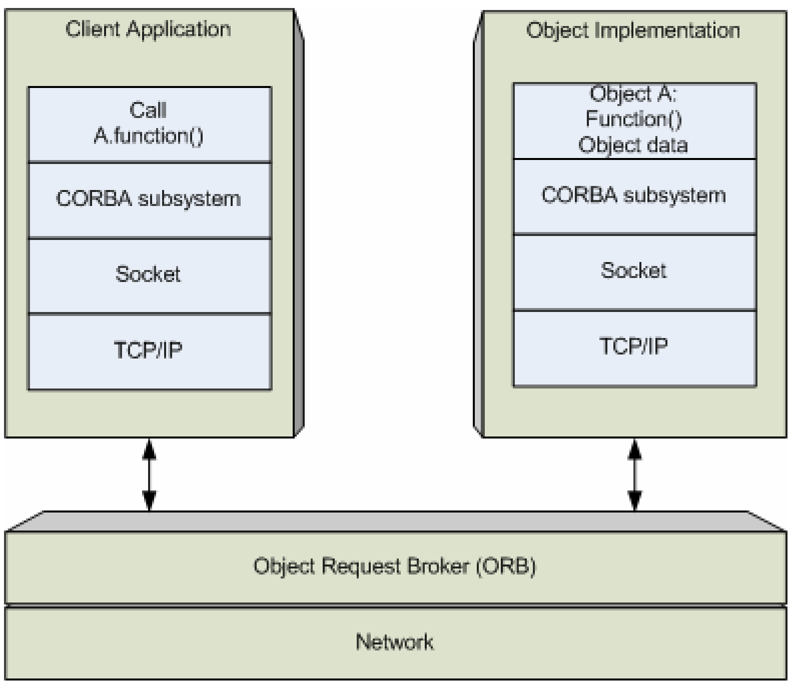


Figure 11: CORBA block diagram

CORBA uses an interface definition language (IDL) to specify the interfaces that objects will present to the world. CORBA then specifies a “mapping” from IDL to a specific implementation language like C++ or Java. This mapping precisely describes how the CORBA data types are to be used in both client and server implementations.

2.2.7 Global Positioning System (GPS)

In our project, positioning data from the sea cages is one of the essential context sources. For this a GPS receiver is used. GPS [16] is a space-based radio-navigation system, consisting of 24 satellites and ground support. The system is operated by the United States military, and provides civilian users with accurate information about their location and velocity anywhere in the world.

Figure 12: The 24 GPS satellites [16].

The satellites transmit signals to equipment on the ground. GPS receivers passively receive satellite signals; they do not transmit. GPS receivers require an unobstructed view of the sky, so they are used only outdoors and they often do not perform well within forested areas or near tall buildings. GPS operations depend on a very accurate time reference, which is provided by atomic clocks on board.

A GPS receiver "knows" the location of the satellites, because that information is included in satellite transmissions. By estimating how far away a satellite is, the receiver also "knows" it is located somewhere on the surface of an imaginary sphere centred at the satellite. It then determines the sizes of several spheres, one for each satellite. The receiver is located where these spheres intersect.

2.2.7.1 NMEA 0183 specification

NMEA stands for The National Marine Electronics Association and is a group dedicated to the education and advancement of the marine electronics industry and the market which it serves. The NMEA 0183 is a combined electrical and data specification for communication between GPS receivers. It uses a simple serial protocol transmitting a "sentence" from one "talker" to one or more "listeners". This NMEA format is transmitted periodically from the GPS serial output in ASCII text.

An example of a NMEA sentence (position and time) [22]:

```
$GPRMC,235947.000,V,0000.0000,N,00000.0000,E,,,041299,,*1D
```

The different fields in the example are described in Table 6.

Table 6: NMEA 0183 position and time fields [21]

Field	Example	Comments
Sentence ID	$GPRMC	
UTC Time	092204.999	hhmmss.sss
Status	A	A = Valid, V = Invalid
Latitude	4250.5589	ddmm.mmmm
N/S Indicator	S	N = North, S = South
Longitude	14718.5084	Dddmm.mmmm
E/W Indicator	E	E = East, W = West
Speed over ground	0.00	Knots
Course over ground	0.00	Degrees
UTC Date	211200	DDMMYY
Magnetic variation		Degrees
Magnetic variation		E = East, W = West
Checksum	*25	
Terminator	CR/LF	

2.2.7.2 The UTM system

The Universal Transverse Mercator (UTM) [23] projection system is a grid-based method of specifying locations on the surface of the Earth. It has the same goal as the traditional latitude and longitude representation, but differs in several respects. The UTM system is not a map projection, but rather employs a series of zones based on specifically defined transverse mercator projections. The system uses two kinds of zones: latitude zones and longitude zones. There exist 20 latitude zones and 60 longitude zones, where the latitude is given by letters and longitude by numbers.

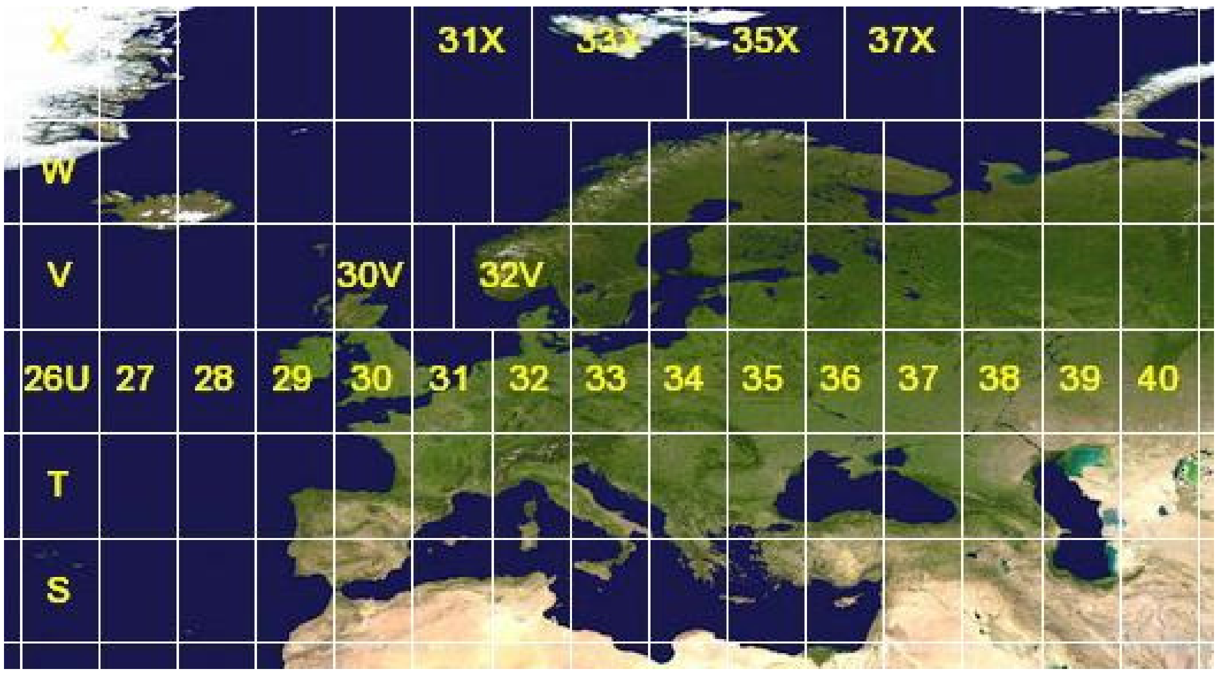


Figure 13: The UTM zones in Europe

To give a location in the UTM system, first the latitude zone is given, then the longitude zone, and in addition numbers of meters east and north of the reference point (bottom right of the zone). By using the UTM system, GPS positions can easily be handled, since the location is given in metres.

2.2.8 GPRS/EDGE

The sensor information from the sea cages is transferred over the air via GPRS [17] (General Packet Radio Services). GPRS is a packet-switched technology that enables Internet Protocol data communications over mobile networks. GPRS offers a tenfold increase in data speed over previous (circuit-switched) technologies, up to 115 kbit/s (in theory). Typical real-world speeds are around 30-40 kbit/s. Like GSM in general, GPRS is an open standards driven system and the standardisation body is the *3GPP* (Third Generation Partnership Project) [18].

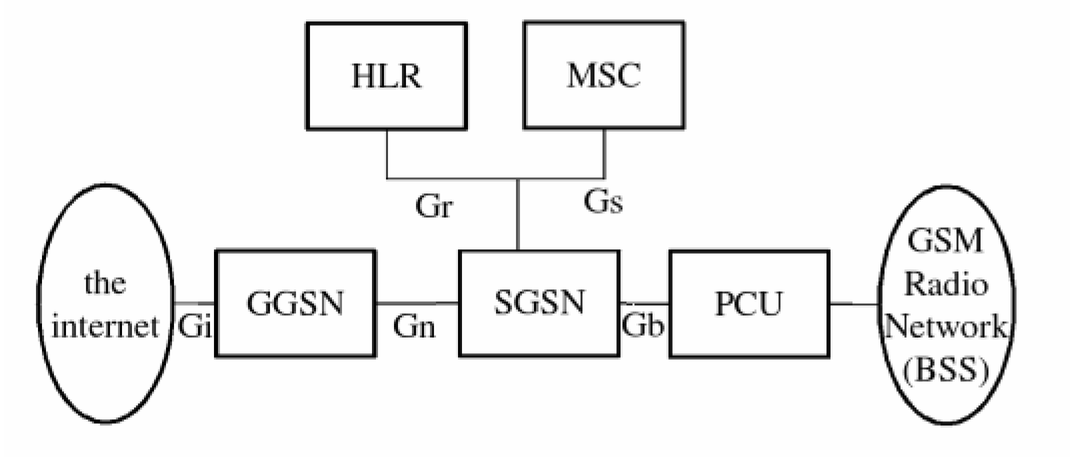


Figure 14: GPRS architecture

The GGSN (Gateway GPRS Support Node) is the node which carries out the role in GPRS equivalent to the Home Agent in Mobile IP. It is a router which de-tunnels user data from GPRS Tunnelling Protocol and sends out normal user data IP packets. The SGSN (Serving GPRS Support Node) is the node which in some sense carries out the same function as the Foreign Agent in Mobile IP.

The Packet Control Unit (PCU) is a late addition to the GSM standard. It performs some of the processing tasks of the BSC, but for packet data. The allocation of channels between voice and data is controlled by the base station, but once a channel is allocated to the PCU, the PCU takes full control over that channel. The PCU can be built into the base station, built into the BSC or even, in some proposed architectures, it can be at the SGSN site

The M2M modules on the sea cages also support EDGE (Enhanced Data for Global Evolution). This is an upgrade for GSM/GPRS networks that triples data rates (speed) over standard GPRS. EDGE is used automatically when both the phone and network support it. EDGE phones will automatically revert to the slower GPRS standard when EDGE service is not available.

3 Realization of FiFaMoS

This chapter will describe the entire development process of the fish farm monitoring system, or FiFaMoS, as it is called. The work has been done according to a model called the "Rational Unified Process", or RUP. Using the RUP, software product lifecycles are broken into individual development cycles, or iterations. These iterations are further broken into several phases. In RUP, these phases are as follows:

- Inception Phase
- Elaboration Phase
- Construction Phase
- Transition Phase

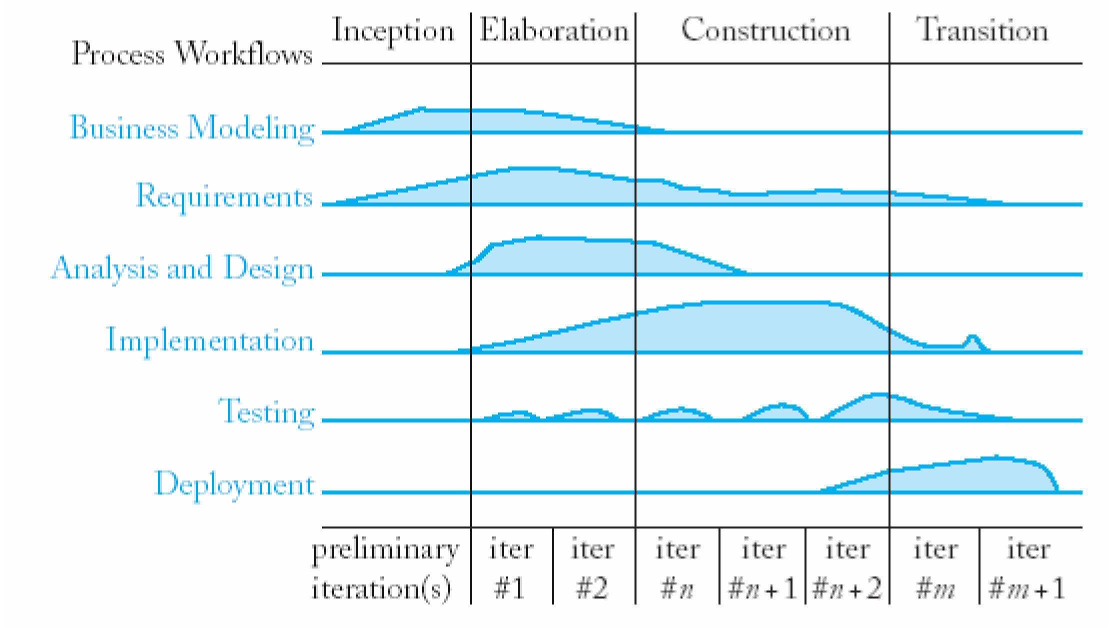


Figure 15: The RUP design process

As seen from the figure, RUP is two-dimensional; the horizontal axis represents time and shows the dynamic aspects of the process model. The vertical axis represents the different aspects of the process in form of different activities.

Due to the limited time of the project, only two iterations were completed, but the software engineering have still benefited from using the RUP model. All the different phases will be documented in this chapter, and detailed description of the work process can be found in Appendix G.

3.1 Prestudy

The prestudy part of the project involves calculating the goals and discovering external conditions, success factors and risks. We have also listed the standards and software used, and the project organization.

3.1.1 Project goals

The project goals are basis for:

- agreeing with the employer on the project result
- project planning and management
- getting a common understanding of the tasks within the project group

Goals are separated into two different groups: The result goals and the effect goals.

3.1.1.1 Result goals

The result goals describe the definitive results of the project. Result goals of this project include the following tasks:

- Specify, design and implement a context-aware service on the APMS context management system.
- The service will be realized by a collaboration of components contained in and interworking with the context manager.
- Specify, design and implement a context source application on a M2M GPRS module
- Specify, design and implement a context consumer client for PCs
- Specify, design and implement a context consumer client for mobile phones
- The project will span over 21 weeks
- The system shall be finished by June 12th

3.1.1.2 Effect goals

The effect goals describe what the employer will achieve by the project. Effect goals of our project include:

- Increased development and knowledge of context-aware applications
- Increased development and knowledge about M2M equipment and applications
- Increased GPRS traffic that leads to increased revenue

In addition, the fish farm industry will have the following effect goals:

- Increased control of the fish farms
- Optimizee the fish production process
- Reduced travelling for the fish farm operators due to increased surveillance, and more efficient work flow
- Reduced fish loss and environmental damage if a sea cage is drifting away

3.1.2 Critical success factors

Many factors will affect the success of the project. The system will include many different hardware types that will need to cooperate, and some of the existing software modules will have to work as we anticipate. The most important critical success factors are identified in the following list:

- The sensor gathering module will have to be able to communicate with the context manager
- The GPS must be able to communicate with the sensor gathering module
- Sensors used must be compatible with the inputs of the sensor gathering module
- The context manager must have enough storage capacity to store at least a month of sensor information
- The context manager must have high performance to handle all requests from context sources and context consumers
- All nodes in the system will have to be stable and operate error free. This escpecially applies to the application that acquires sensor information, because debugging and error recovery is difficult off-shore.
- If an error occurs on an off-shore node, it should be fixed by an automatic reboot.
- The clients must be easy to operate and have a user-friendly graphical interface.

3.1.3 Risk analysis

The risk analysis documents conditions that can prevent the project from succeeding. First some main elements of risks in this project are mentioned, and then they are plotted in a diagram showing the degree of consequence and probability. Elements of risk top right will have to be carefully investigated before the project can continue. An investigation of the elements follows after the diagram.

1. The M2M technology used is new and not that much tried and tested.
2. The APMS context manager is under development, and bugs can prevent our system from working properly
3. Needed expertise and resources can be difficult to obtain

4. The system implementation can be so unstable that the uptime for the system will not be good enough
5. Low system security will prevent the system from being useful
6. The M2M module does not have enough functionality to manage all the tasks mentioned in the requirements

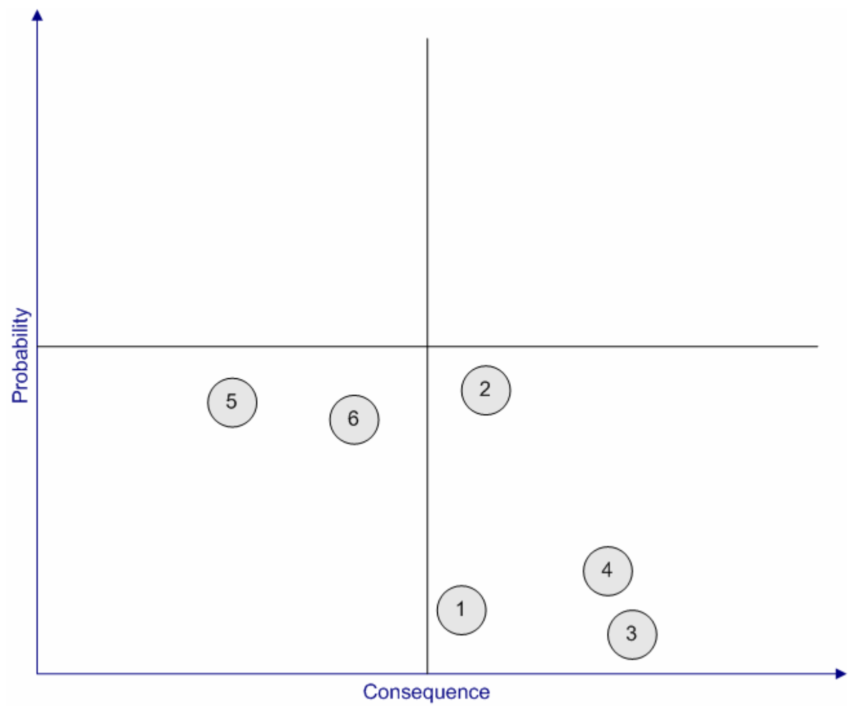


Figure 16: Risk analysis diagram

3.1.3.1 Comments

The first element of risk is that the M2M technology is new and not that much tried and tested. This element has low probability and medium consequence. Even though M2M technology is not that much used in public, it has been around for several years, and a lot of research has been done in the field. It should be possible to find an M2M product that is of high enough quality to serve as a sensor controller on a sea cage. If not, an industrial computer with GPRS interface would be a good replacement, but that would cost more, and add a lot of complexity to the system.

The second element of risk claims that the APMS middleware is under development, and bugs can prevent the system from working properly. This is the element with the highest probability, and it has got a medium consequence level. The middleware is not yet 100% finished, but shall have enough functionality for the project to be useful. If critical bugs stop the system from functioning, it is a serious matter, but since contact with the development team is established, bugs can be fixed during the project. If very serious bugs occur, it is possible to use other context management systems, like Akogrimo [1] for instance.

The third element of risk says that it can be difficult to obtain the needed expertise and resources. This element has low probability and high consequences. Since the project work will be located at Telenor Research and Development, highly skilled personnel, and a high amount of technical equipment will be accessible. If further expertise or equipment is needed, it could probably be found on the Internet.

The fourth element of risk mentions that a bad system implementation could make the system uptime too low. This element has low probability and medium consequences. After several years with programming experience, the development team should be capable of delivering a system that is stable enough to operate over a long time. If stability problems should occur, it is not the biggest problem. The main goal of the thesis is to make a functioning prototype, not an implementation that works error-free for years.

The fifth element of risk claims that low system security will prevent the system from being useful. This element has medium probability and low consequences. High security is important in a finished product ready for the market, but in this prototype, it will not be of high interest.

The last element of risk is that the M2M module does not have enough functionality to manage all the tasks mentioned in the requirements. This element has medium probability and medium consequences. If this happens, other hardware modules will have to be added, or the M2M module will have to be replaced by an industrial computer. If this is not possible, the requirements list will have to be re-evaluated.

Seen as a whole, the project seems to have a tolerable amount of risk, and can be continued.

3.1.4 Standards and software used

In this project, the implementation is done in Java, both Java Standard Edition and Java Micro Edition. To ease the implementation, software development tools are used. Eclipse was chosen due to earlier positive experiences with this software. When implementing the FiFaMoS Mobile Context Consumer and the FiFaMoS Context Source application, the EclipseME plug-in was used. This plug-in makes development of J2ME MIDlets easier. EclipseME does the demanding work of connecting wireless toolkits to the Eclipse development environment, and allows the user to focus on the development process.

The wireless toolkit used is Sun Java Wireless Toolkit 2.1. This is a toolbox for developing wireless applications that are based on J2ME's Connected Limited Device Configuration (CLDC) and Mobile Information Device Profile (MIDP), and designed to run on cell phones, and other small devices. The toolkit includes emulation environments, performance optimization and tuning features that improve the development process.

To configure the sensor value gathering M2M module, Aplicom N12i configurator is used (see appendix E.1). With this software, GPRS settings, GSM settings, serial port settings, and other parameters can be altered. It is also the software that is used to upload and start applications on the M2M module.

To test M2M applications without uploading them to the module, the Aplicom N12i concept simulator is used (see appendix E.2). This simulator has the same functionality as the module, and also logs debug information.

The context manager runs on a computer situated at the PATS-lab at Telenor Tyholt. To avoid visiting this computer every time new configuration is needed, a desktop sharing system is used. The software used is Real VNC Viewer 4. To upload files to the context manager, SSH Secure Shell File Transfer is used.

When creating the service on the APMS context manager, a tool called JMX monitor was used (see appendix E.3). This makes it easy to invoke methods and test the service. In addition it provides good debug information that makes error searching easier.

All reporting is done in Microsoft Office Word 2003, and figures are either created in Microsoft Visio or edited in Jasc Paint Shop Pro 9.

3.1.5 Project organization

The project is being worked out by Frank Paaske and Jon Arne Grødal. We have been doing this project together, but our focus has been on different parts of the system, Frank is responsible for the sensor value gathering system, and the context management, and Jon Arne is responsible for the user side of the system. Also several others are taking part in the project, including Per-Oddvar Osland and Frode Flægstad at Telenor Research and Development which are supervisors. The APMS context manager used has been developed by Arne Munch-Ellingsen, and he is helping us with the development of the components needed in our system. He has also been updating the context manager after requests from us.

3.2 Requirement specifications

In this chapter, the functions of the system will be described. Requirements for implementaion, data and system quality will also be discussed.

In a fish farm there are several needs for surveillance. Each sea cage will have several sensors, and a GPS for positioning info. The status of the sensors is sent periodically to the context manager which distributes this information to the receivers. There exist many types of sensors that can be used in a fish farm. Examples are:

- Positioning (GPS)
- Sea cage net sensor
- Wave sensor
- Water level sensor
- Water quality sensor
- Temperature sensor
- Wind sensor
- Food level sensor
- Current sensor
- pH sensor
- Oxygen content

Since this is only a prototype, not all sensor types will be implemented in the system, but the system shall provide an easy way of adding new sensor types. We are not going to test the system on a real sea cage, and have therefore only implemented support for some sensors. During testing, dummy sensors will fill the role of producing sensor values, but the positioning information will come from a real GPS receiver.

3.2.1 Functional requirements

Table 7 lists the tasks that the system is required to perform. The phase column states if the task is going to be implemented in iteration one or iteration two.

Table 7: Functional requirements

Requirement	Description	Phase
FR1	Read sea cage sensor values	1
FR1.1	Read sea cage position (GPS)	2
FR1.2	Read sea cage net sensor	1
FR1.3	Read wave sensor	1
FR1.4	Read water level sensor	1
FR1.5	Read temperature sensor	1
FR1.6	Read water quality sensor	1
FR1.7	Read food level sensor	1
FR1.8	Read wind sensor	1
FR1.9	Read current sensor	1
FR1.10	Read pH sensor	1
FR1.11	Read oxygen content	1
FR2	Get historical sensor values (all sampled values)	1
FR2.1	Get history for last 60 minutes	1
FR2.2	Get history for last 24 hours	1
FR2.3	Get history for last week	1
FR2.4	Get history for last month	1
FR3	Send alarm	2
FR3.1	Send alarm when sensor values are out of range	2
FR3.2	Send alarm when a sea cage is out of position	2
FR3.3	Send alarm via SMS/MMS	2
FR4	Display alarm in application/web interface	2
FR5	Start surveillance camera session	2
FR6	Store logging data to file	2
FR7	Alter fish farm configuration from the PC client	2
FR7.1	Alter frame configuration	2
FR7.1.1	Add new frame	2
FR7.1.2	Update existing frame	2
FR7.1.3	Delete existing frame	2
FR7.2	Alter sea cage configuration	2

Requirement	Description	Phase
FR7.2.1	Add new sea cage	2
FR7.2.2	Update existing sea cage	2
FR7.2.3	Delete existing sea cage	2
FR7.3	Alter sensor configuration	2
FR7.3.1	Add new sensor	2
FR7.3.2	Update existing sensor	2
FR7.3.3	Delete existing sensor	2

3.2.2 Use case modelling

Use case modelling captures the potential requirements of the system. Each use case provides one or more scenarios that convey how the system should interact with the end user or another system to achieve a specific business goal. The following section describes the most important use cases for the system. The graphical use case diagram shows the functions, and textual descriptions come afterwards.

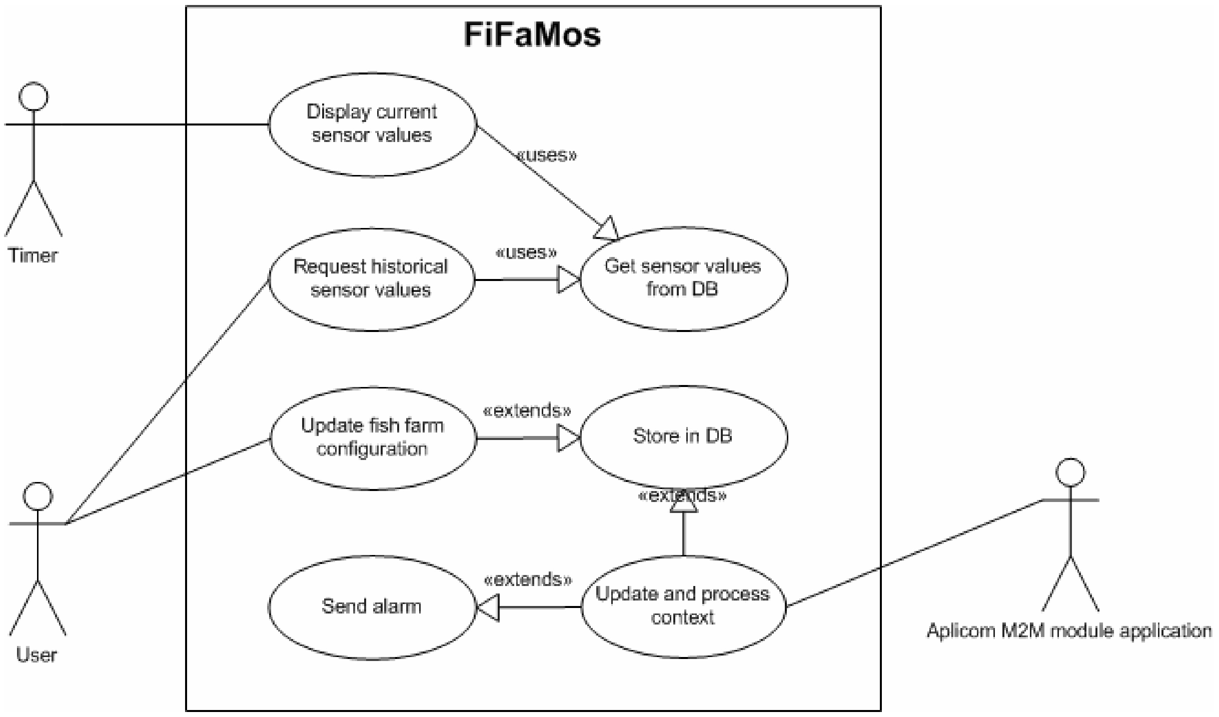

FiFaMos
Timer
User
Aplicom M2M module application
Display current sensor values
«uses»
Request historical sensor values
«uses»
Get sensor values from DB
Update fish farm configuration
«extends»
Store in DB
«extends»
Send alarm
«extends»
Update and process context

Figure 17: Identified use cases in the context consumer client

In the FiFaMoS system, three main actors are identified: The end user, and the Aplicom M2M module, and a timer. The system functionality could also be described by defining other actors (like for instance the sensors), or with one diagram per application (client, context manager and M2M module application), but it is found that this description represents the functionality in the best way. It is also the timer, user and the M2M module application that triggers most of the functionality of the system.

The diagram introduces two kinds of relations: the << extends >> relation, and the << uses >> relation. This first indicates that the behaviour of the extension use case may be inserted in the extended use case under some conditions, and the latter that the use case often depends on the outcome of the included use case. The << uses >> relation is also referred to as the << includes >> relation in some documents. Lines from the actors to the use cases indicate that the use case is triggered by the actor.

3.2.2.1 Use-Case Specification: <Use-Case: Update and process context>

Table 8: Use-Case: Update and process context

Actors:	Aplicom M2M module application
Description:	This use case makes the M2M application able to update context information in the context manager. The context information is received and processed. The processing involves interpreting mV sensor values to real values like degrees (temperature) or percentage value (oxygen level).
Preconditions:	Aplicom M2M module application running, context manager running.
Postconditions:	Processed context are stored in the database and required alarms are sent
Normal Flow:	1. Aplicom M2M module application sends new context information 2. Context is received 3. Context is interpreted 4. Sensor data is compared to the preset limits 5. The value is allowed
Alternative Flows:	1. Aplicom M2M module application sends new context information 2. Context is received 3. Context is interpreted 4. Sensor data is compared to the preset limits 5. The value is not allowed. 6. The alarm use case is triggered
Uses:	
Priority:	High

Frequency of Use:	Every time new context is received from context sources
Business Rules:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

3.2.2.2 Use-Case Specification: <Use-Case: Display current sensor values>

Table 9: Use-Case: Display current sensor values

Actors:	Timer
Description:	This use case makes the user able to view the current sensor values. The sensor values will be displayed in the graphical user interface of the client.
Preconditions:	Context manager running, client (FiFaMoS Context Consumer or FiFaMoS Mobile Context Consumer) running. Database has sensor values stored.
Postconditions:	Client GUI is updated with the current sensor information
Normal Flow:	1. A timer triggers the display current sensor values use case 2. The last sensor values stored in the database is requested and sent back to the client 3. Sensor values are displayed in the client GUIs.
Alternative Flows:	
Uses:	Get sensor values from DB
Priority:	High
Frequency of Use:	Every time new context is received from context sources
Business Rules:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

3.2.2.3 Use-Case Specification: <Use-Case: Store in DB>

Table 10: Use-Case: Store in DB

Actors:	Aplicom M2M module application or user
Description:	This use case stores the context (sensor values) to a database, so that it can be fetched later by the clients. This makes it possible to view historical sensor data.
Preconditions:	Context manager running, Aplicom M2M module application or PC client running.
Postconditions:	Sensor values or new configurations are stored in the database
Normal Flow:	1. New interpreted context from the update and process context use case is received 2. Sensor values are stored in the database
Alternative Flows:	1. New configuration is received from the user 2. Configuration is stored in the database
Uses:	
Priority:	High
Frequency of Use:	Every time new context is received from context sources , or new configuration is received from the context consumers.
Business Rules:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

3.2.2.4 Use-Case Specification: <Use-Case: Request historical sensor values>

Table 11: Request historical sensor values

Actors:	User
Description:	This use case makes the user able to view the historical sensor values. The historical sensor values will be displayed in a graph.
Preconditions:	Context manager running, client (FiFaMoS Context Consumer

	or FiFaMoS Mobile Context Consumer) running.
Postconditions:	Sensor values are displayed in a graph in the client
Normal Flow:	1. The user chooses to view historical sensor values (which sensor and the period) 2. The context manager gets the values from the DB, and displays them in the UI.
Alternative Flows:	
Uses:	Get sensor values from DB
Priority:	High
Frequency of Use:	Every time the user chooses to view historical sensor values.
Business Rules:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

3.2.2.5 Use-Case Specification: <Use-Case: Get sensor values from DB>

Table 12: Use-Case: Get sensor values from DB

Actors:	User or timer
Description:	This use case makes is possible to get stored sensor values from the database in the context manager.
Preconditions:	Context manager running, client (FiFaMoS Context Consumer or FiFaMoS Mobile Context Consumer) running.
Postconditions:	
Normal Flow:	1. The user chooses to view historical sensor values (which sensors and the period) 2. The context manager gets the values from the DB, and displays them in the UI.
Alternative Flows:	1. The timer task requests the current sensor values 2. The context manager gets the values from the DB, and

	displays them in the UI.
Uses:	
Priority:	High
Frequency of Use:	Every time the user chooses to view historical sensor values, or the timer task requests the current values.
Business Rules:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

3.2.2.6 Use-Case Specification: <Use-Case: Send alarm>

Table 13: Use-Case: Send alarm

Actors:	Aplicom M2M module application
Description:	This use case sends an alarm to the user if a sensor value is out of bounds
Preconditions:	Context manager running, PC client running or operator mobile phone reachable
Postconditions:	
Normal Flow:	1. A sensor value is out of bounds 2. An alarm message is sent to UI
Alternative Flows:	1. A sensor value is out of bounds 2. An alarm message is sent to the operators mobile phone via SMS
Uses:	
Priority:	High
Frequency of Use:	Every time a sensor value is out of bounds
Special Requirements:	
Assumptions:	
Notes and Issues:	

3.2.2.7 Use-Case Specification: <Use-Case: Update fish farm configuration>

Table 14: Use-Case: Update fish farm configuration

Actors:	User
Description:	This use case lets the user alter the fish farm configuration. It is possible to add/remove/modify frames/sea cages/sensors.
Preconditions:	Context manager running, PC client running
Postconditions:	New configuration is stored in the database, and the involved M2M modules are automatically updated
Normal Flow:	1. A user alters the configuration with the PC client 2. The configuration is sent to the context manager 3. The context manager alters the database according to the new configuration.
Alternative Flows:	
Uses:	
Priority:	Medium
Frequency of Use:	Every time a user wants to alter the fish farm configuration
Business Rules:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

3.2.3 Implementation requirements

This section describes various implementation requirements. That includes language and technological choices as well as requirements to different implementation solutions.

Table 15: Implementation requirements

Requirement	Description	Priority
IR1	The system shall use the APMS context management system	High
IR2	Output/input/processing/configuration plug-ins shall be implemented in java/xml	High

Requirement	Description	Priority
IR3	The mobile client shall be implemented in J2ME	High
IR4	The PC client shall be implemented in J2SE	High
IR5	The M2M module application shall be implemented in J2ME	High
IR6	XML-RPC web services shall be used for communication between the mobile client and the context manager	High
IR7	XML-RPC web services shall be used for communication between the PC client and the context manager	High
IR8	XML-RPC web services shall be used for communication between the M2M module and the context manager	High
IR9	Objects shall be represented in XML when transferred via XML-RPC	High
IR10	The M2M module configuration is set by a XML document sent in the web service response.	Medium
IR11	Alarms shall be triggered by the context manager	High
IR12	Alarms shall be sent via XML-RPC to the PC client, and via SMS to the operators mobile phone	Medium
IR13	The M2M module shall send the SMS alarm	Medium
IR14	Camera sessions media is transferred via the context manager	Medium
IR15	The communication link between the M2M module and the context manager shall be GPRS/EDGE	Medium

3.2.4 Data requirements

To allow the user to view historical sensor data, sensor context has to be stored in persistent memory. For this, a database in shall be used. Over time, a lot of data will be generated, and the database will have to be able to handle these amounts of data. Below, a scenario where a fish farm containing 50 sea cages, each with five sensors, are inspected. The sea cages are set up to poll their sensor information once a minute.

According to the normal nature of typical sensor information, each value (including timestamp and various other information) will need approximately 100 bytes of information stored in the database. For the history function to be useful, at least one month of data should be stored. This generates the following amount of data:

*Number of sensors per sea cage * number of sea cages per frame * number of frames * amount of data per sensor value * number of minutes in a month = 5 sensors * 10 sea cages * 5 frames * 100 bytes * (60 * 24 * 30) minutes = 1080000000 bytes/month = 1080000 KB/month = 1080 MB/month = 1GB/month.*

This shows that to get one month of sensor history in a fish farm with 50 sea cages, the data base will have to be able to cope with approximately 1GB of data. Most of today's computers have this much storage capacity, and most of today's databases will have no problems handling this much information.

To make the configuration of the the mobile client application stay intact between sessions and during power loss or reboot, it has to be stored permanently. It is assumed that the configuration will need about 200 bytes of storage place, and that can be placed in the module's or phone's internal record store (persistent memory).

The PC client will also have to store its settings permanently. The size of the configuration is about the same as the mobile client, and the settings can be stored in a properties file on the hard disk.

Not all data transmission in the system uses the Internet. The GPS and the camera are connected to the M2M module via serial interfaces. These interfaces are going to be configured to use a baud rate at 115200 bps. The GPS sends the positioning data as pure ASCII text according to the NMEA 0183 standard (see section 2.2.7.1), while the camera sends and receives hexadecimal commands and data.

3.2.5 Quality requirements

The system is supposed to offer near real-time responses. However, low bandwidth links like GPRS, and multi hop links like the Internet will add a considerable amount of delay. This delay should not be more than a second, but for responses with high amount of data (like the get historical sensor values functionality) the response time for the mobile client could be as high as a couple of seconds. This should not be a problem on the PC client (assuming the PC has got a broadband connection).

It is important that the sensor values presented in the clients are up to date. If sensor data shall be pushed periodically to the context manager, and the clients shall pull the sensor data periodically, the values could be aged. In a worst case scenario the sensor value displayed in the client is not the current value, but a value with the age: sensor gathering interval + client refresh interval + transmission and processing delay. If both these interval values are set to one minute, the sensor values displayed in the client could be up to 2 minutes old. However, they will most often not be that old, and our assumption is that about 1 minute will be a normal age for the displayed sensor value. This is considered to be sufficient for this kind of surveillance. If not, the refresh intervals will have to be set lower. However, this increases the demands for data storage, and the GPRS expenses.

Since the M2M module(s) will be placed off-shore, error recovery and service will not be an easy task. Therefore it is important that the application is fail-safe. However, this is not always easy to accomplish, so error recovery routines must be implemented. It is wanted that the M2M modules shall reboot themselves if an error occurs. The FiFaMoS system shall not fail more often than once in a 150 days period, and error recovery shall not take more than two hours.

To sum up the chapter, the following quantifiable quality requirements were settled:

Table 16: Quality requirements

Requirement	Description
QR1	MTBF: The system shall have a mean time between failures of at least 150 days.
QR2	MTTR: The system shall have a mean time to repair of less than two hours.
QR3	Response times for getting current sensor data for a sea cage shall not be over one second
QR4	Displayed sensor value age shall not be over two minutes

Assuming these requirements, the system will have an availability *A* like this:

$$A = MUT/MTBF = 1 – U = 1 – MTTR/MTBF = 1 – (2/150*24) = 0{,}9994444444 = 99{,}9444\%$$

3.2.6 Technical requirements

Technical requirements specify the high-level requirements to the hardware and software of the system. The technical requirements are stated in Table 17.

Table 17: Technical requirements

Requirement	Type	Description
TR1	Software	The system shall not be dependent of program modules not supplied
TR2	Software	The clients shall be able to run on multiple operating systems, including Windows and Unix/Linux
TR3	Hardware	The system shall be accessible from any normal PC/PDA/other terminal
TR4	Hardware	The system has to be run hardware that satisfies the requirement to response time

3.3 Design

In the design phase of the project, various system models are made. These models are represented graphically in diagrams. Most of the diagrams used in this project are UML diagrams, but also SDL is used. The system model contains of three main parts:

- The functional model that shows the functionality from the user's point of view. This includes Use Case Diagrams (introduced in the requirement specification).
- The object model, that shows the structure of the system using objects, attributes, operations, and associations. This includes class diagrams (see Appendix F).
- And at last the dynamic model that shows the internal behaviour of the system. That includes Sequence Diagrams and Activity Diagrams.

In addition to the UML and SDL modelling, the architecture and context data model are also defined. First the architecture of the system is described.

3.3.1 Architecture description

This section describes the system architecture. In the architecture design, the main approach was to keep the number of nodes as low as possible. This was done to reduce complexity and the number of failure sources.

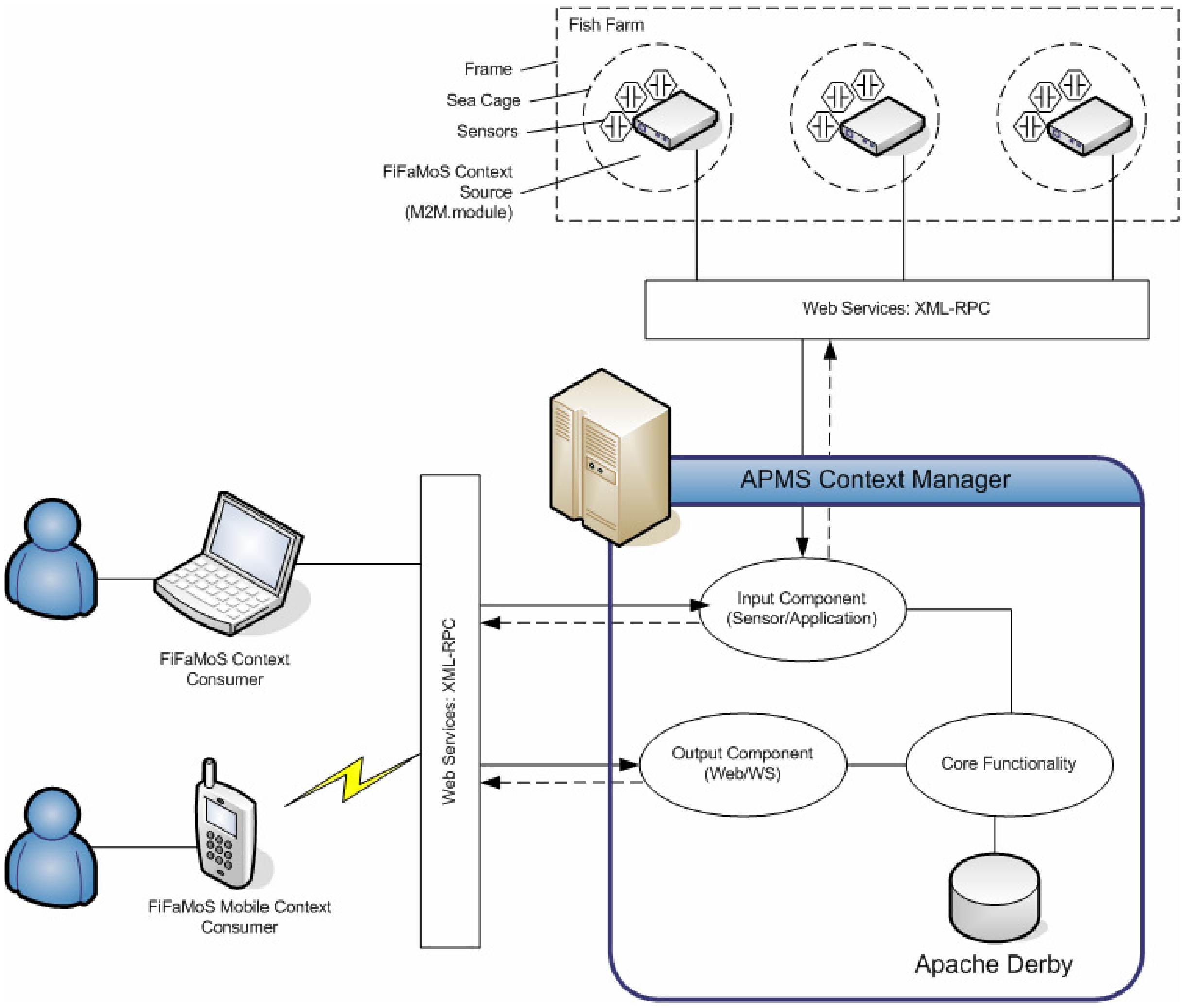


Figure 18: The system architecture

The whole system consists of three main parts as suggested by Indulska in Figure 5: The context source part (FiFaMoS Context Source), the context manager (APMS middleware) and the context consumers (FiFaMoS Context Consumer and FiFaMoS Mobile Context Consumer). The interaction between the nodes is suggested using web services. This is done to make the system as loosely coupled as possible, and to get a Service Oriented Architecture (SOA). Because the only web server in the system is the one in the APMS middleware, the APMS cannot be the one taking initiative to communication. This way web service calls are made from the context source to push sensor data to the context manager, and equivalent from the context consumer to pull information from the context manager.

The internal architecture of the APMS context manager consists of components. In this system, the only components needed are an input component and an output component. The input component is responsible for receiving data (for instance sensor values) and storing it to persistent memory, while the output component is responsible for sending out requested data.

To get further overview of the system architecture, Figure 19 introduces a block diagram of the system modules and interfaces.

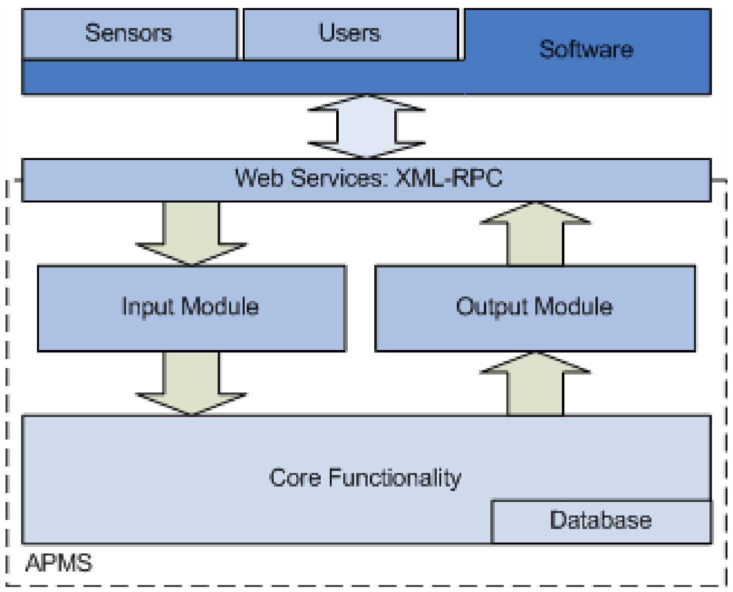


Figure 19: Block diagram of the system architecture

3.3.2 Component view

Figure 20 shows a component diagram of the whole system. This diagram's purpose is to show the structural relationships between the components of the system.

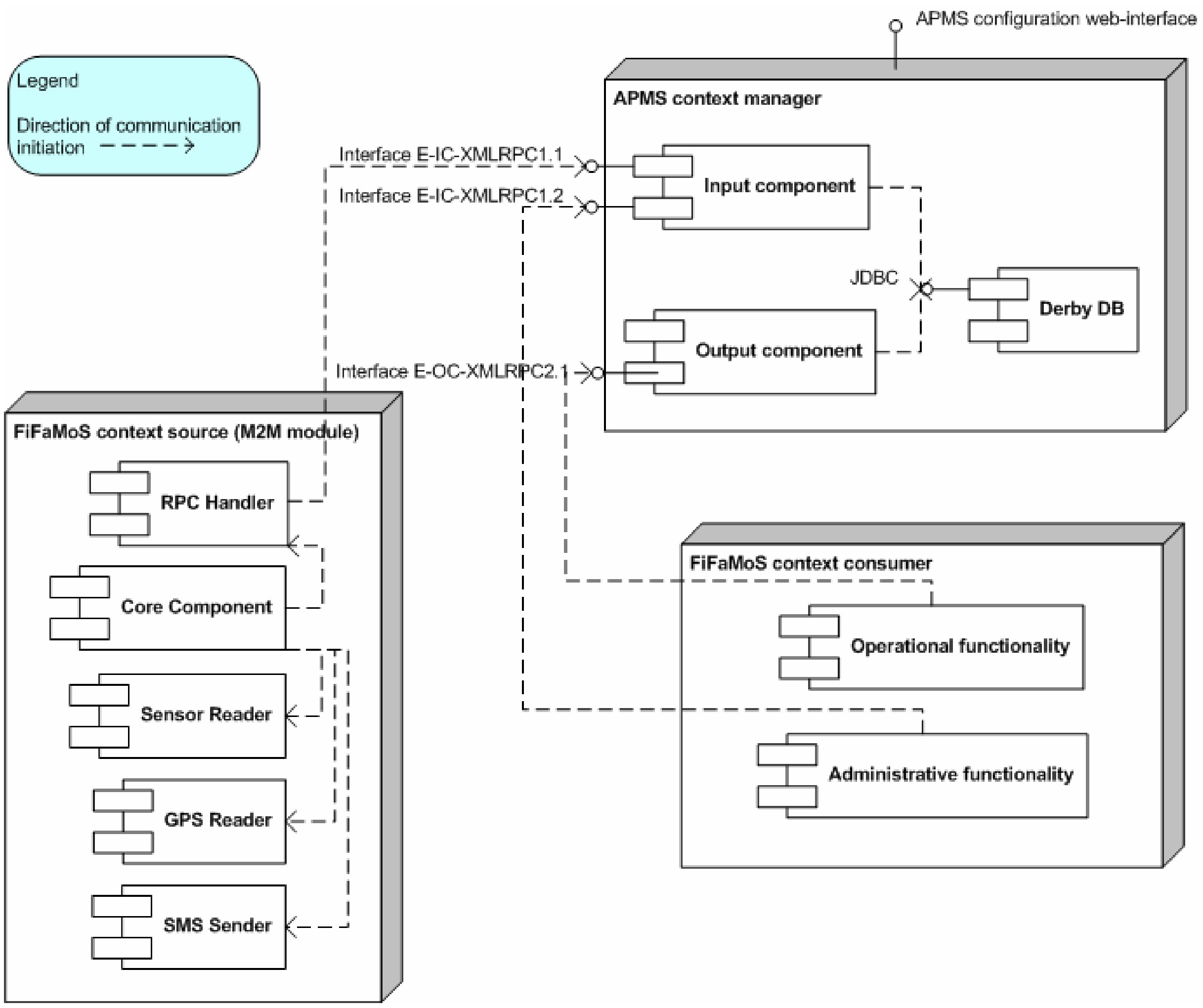


Figure 20: UML component diagram

3.3.3 Database model

The context information in the system is stored permanently to a database. The modelling of the context data was done using an ER-diagram (Figure 21). The different tables are described in detail under:

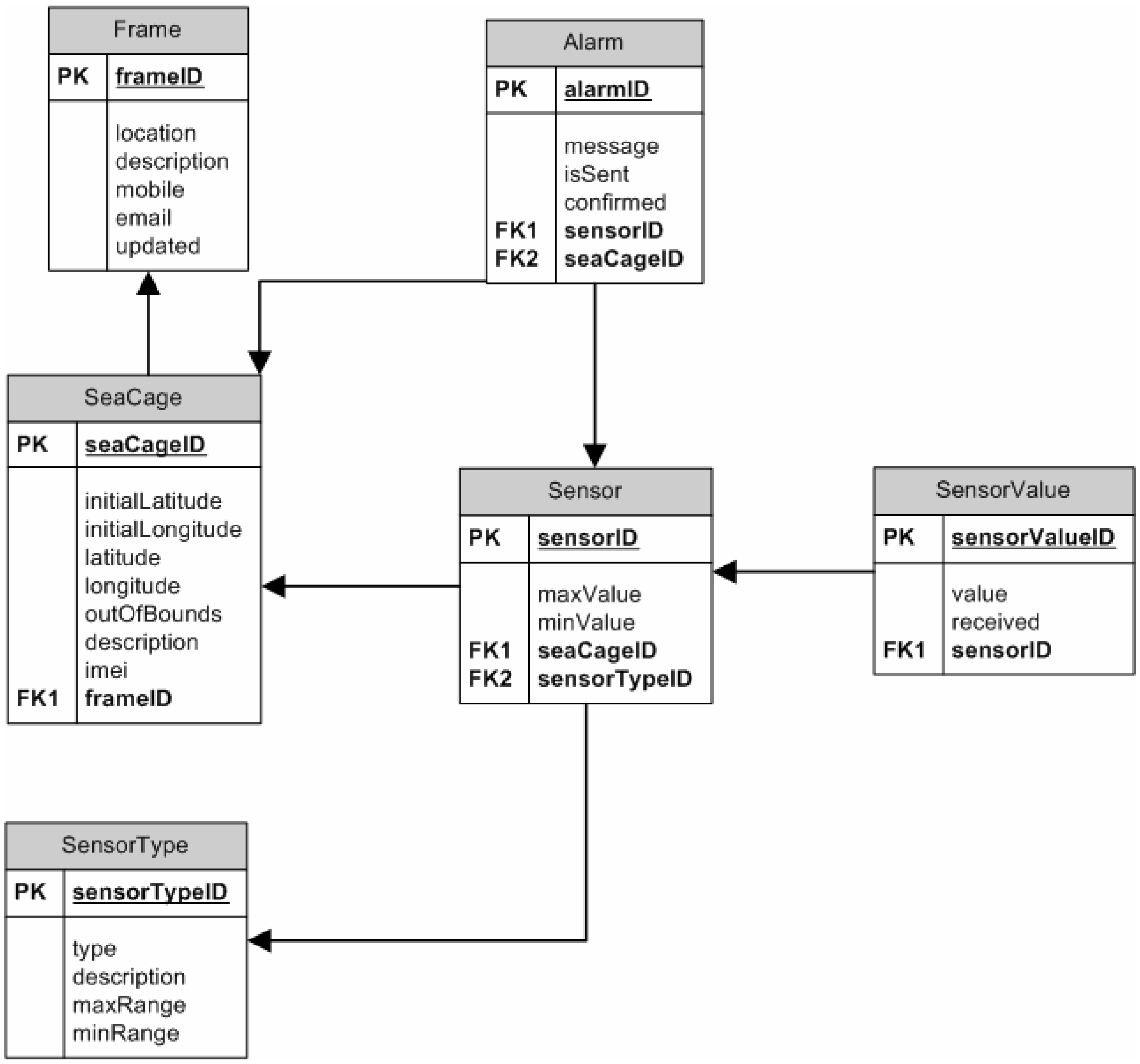


Figure 21: ER data model

3.3.3.1 Frame table

A number of sea cages are tied together in a frame (shown in Figure 1). One frame normally contains 2x5 sea cages, and it is found reasonable to represent these frames in the database to ease navigation between the sea cages.

Table 18: Frame table elements

Name	Description	Source	Attribute type	Attribute domain	Key
FrameID	Unique table identifier	Automatically generated	Integer	Starts with 1, increments by 1	Primary
Location	Textual description of the fish farm location	Set by user	String	Up to 200 characters	None
Description	Frame information	Set by user	String	Up to 200 characters	None
Mobile	Mobile number of the responsible operator	Set by user	String	Up to 200 characters	None
E-mail	E-mail address to the responsible operator	Set by user	String	Up to 200 characters	None
Updated	Indicates if the frame or any of it’s sea cages has been updated	Automatically set when something is updated	Integer	0 or 1 representing true or false	None

3.3.3.2 Sea Cage table

The sea cage is the structure where the fish lives and grows (shown in Figure 1). It is composed by a floating elements (often round) and the net that prevents the fish from escaping. Each sea cage has a set of sensors, and its own M2M module that collects sensor information. The module is identified by the IMEI-field in the table.

Table 19: Sea Cage table elements

Name	**Description**	**Source**	**Attribute type**	**Attribute domain**	**Key**
SeaCageID	Unique table identifier	Automatically generated	Integer	Starts with 1, increments by 1	Primary
InitialLatitude	Where the Sea Cage is located when initialized	GPS data from Aplicom L4002	String	Up to 200 characters	None
InitialLongitude	Where the Sea Cage is located when initialized	GPS data from Aplicom L4002	String	Up to 200 characters	None
Latitude	Where the Sea Cage currently is located	GPS data from Aplicom L4002	String	Up to 200 characters	None
Longitude	Where the Sea Cage currently is located	GPS data from Aplicom L4002	String	Up to 200 characters	None
Description	Sea cage information	Set by user	String	Up to 200 characters	None
IMEI	IMEI number of the M2M module on the sea cage	Set by user	String	Up to 200 characters	None
FrameID	Id of the frame where the sea cage is located	Set by user	Integer	Any of the previously set FrameIDs in the Frame table	Foreign Key

3.3.3.3 Sensor table

The sensor table stores the static information about the sensors. This includes identification, type, maximum allowed value and minimum allowed value.

Table 20: Sensor table elements

Name	Description	Source	Attribute type	Attribute domain	Key
SensorID	Unique table identifier	Automatically generated	Integer	Starts with 1, increments by 1	Primary
SensorTypeID	Indicates the type of sensor from the SensorType table	Set by user	String	Any of the previously set SensorTypeIDs in the SensorType table	Foreign Key
MaxValue	Indicates the upper limit of allowed values	Set by user	Integer	Any integer between the range of the sensor type	None
MinValue	Indicates the lower limit of allowed values	Set by user	Integer	Any integer between the range of the sensor type	None
SeaCageID	Id of the sea cage where the sensor is located	Set by user	Integer	Any of the previously set SeaCageIDs in the SeaCage table	Foreign Key

3.3.3.4 Sensor value table

To allow historical sensor values, each sensor value will have to be stored. This is done in the sensor value table. Each sensor value is stored with a timestamp (received), ID and which sensor the value came from.

Table 21: FishFarm table elements

Name	Description	Source	Attribute type	Attribute domain	Key
SensorValueID	Unique table identifier	Automatically generated	Integer	Starts with 1, increments by 1	Primary
Value	Processed value from the sensor	Physical sensor data from Aplicom 12	Integer	Any value within the range of the sensor type	None
Received	When the sensor value is received	Automatically set	Timestamp	yyyy-mm-dd hh[:mm[:ss[.nnnnnn]]]	None
SensorID	Id of the sensor where the sensor value belongs to	Set by user	Integer	Any of the previously set SensorIDs in the Sensor table	Foreign Key

3.3.3.5 Sensor type table

There exist a number of sensor types. These can be temperature, net growth, oxygen level and many others. The sensor types with accompanying parameters are stored in this table.

Table 22: FishFarm table elements

Name	Description	Source	Attribute type	Attribute domain	Key
SensorTypeID	Unique table identifier	Automatically generated	Integer	Starts with 1, increments by 1	Primary
Type	Name of the sensor type	Set by user	String	Up to 200 characters	None
Description	Textual description of the sensor type	Set by user	String	Up to 200 characters	None
MaxRange	The maximum	Set by user	Integer	+/-	None

Name	Description	Source	Attribute type	Attribute domain	Key
	sensor value outputted from the sensor			2.147.483.648	
MinRange	The maximum sensor value outputted from the sensor	Set by user	Integer	+/- 2.147.483.648	None

3.3.3.6 Alarm table

When a sensor value is out of bounds an alarm is triggered. Every alarm gets stored in the database in the alarm table.

Table 23: FishFarm table elements

Name	Description	Source	Attribute type	Attribute domain	Key
AlarmID	Unique table identifier	Automatically generated	Integer	Starts with 1, increments by 1	Primary
Message	String representing the error message	Set by the context manager	String	Up to 200 characters	None
SeaCageID	Tells which sea cage triggered the alarm.	Set by the context manager	Integer	One of the previously set SeaCageIDs	Foreign key
SensorID	ID of the sensor that triggered the alarm	Set by the context manager	String	Up to 200 characters	Foreign key
Confirmed	Indicates if and when the alarm is confirmed	Set by client when user confirms	Timestamp	yyyy-mm-dd hh[:mm[:ss[.nnnnnn]]]	None
isSent	Indicates if the alarm is sent or not	Set by the context manager	Integer	0 or 1 representing true or false	None

3.3.4 Interfaces

This section describes the interfaces between the components introduced in Figure 20. The most important message flows in the system are shown, and they are presented as UML sequence diagrams.

3.3.4.1 FiFaMoS Context Consumer – APMS context manager (E-OC-XMLRPC2.1)

When the client starts, it needs info about the frames and sea cages in the fish farm. The client contacts the context manager, which returns this information as XML. The XML is interpreted, and the FiFaMoS Context Consumer GUI is updated with this information.

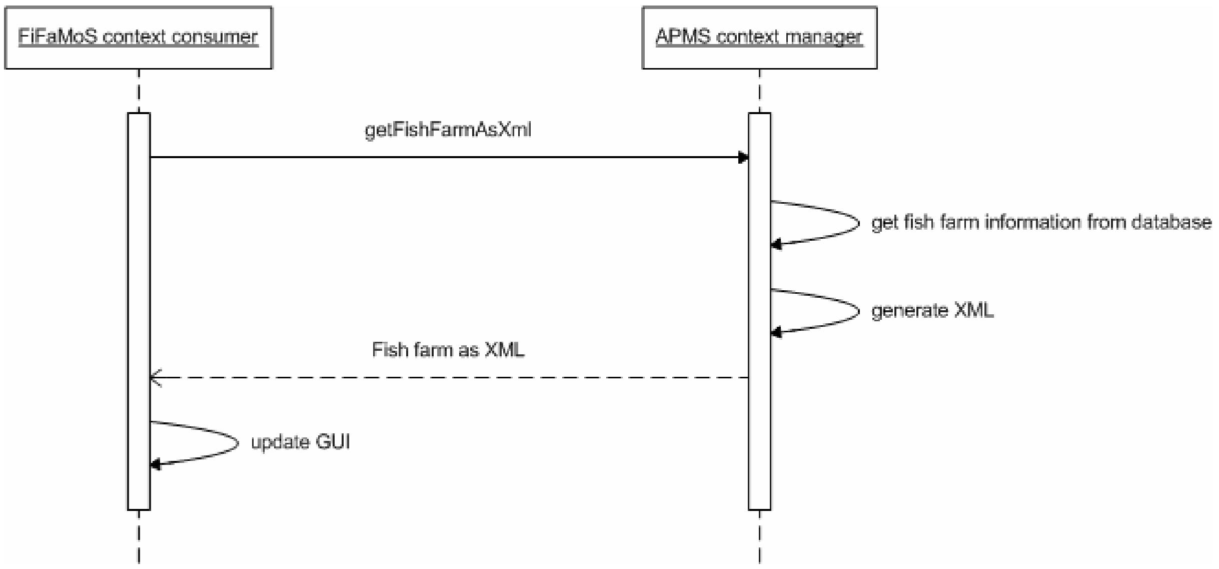


Figure 22: Initialize sequence diagram

When a user selects the sea cage that he/she wants to view information on, a new timer task is created. This timer task gets the current sensor information periodically for the chosen sea cage.

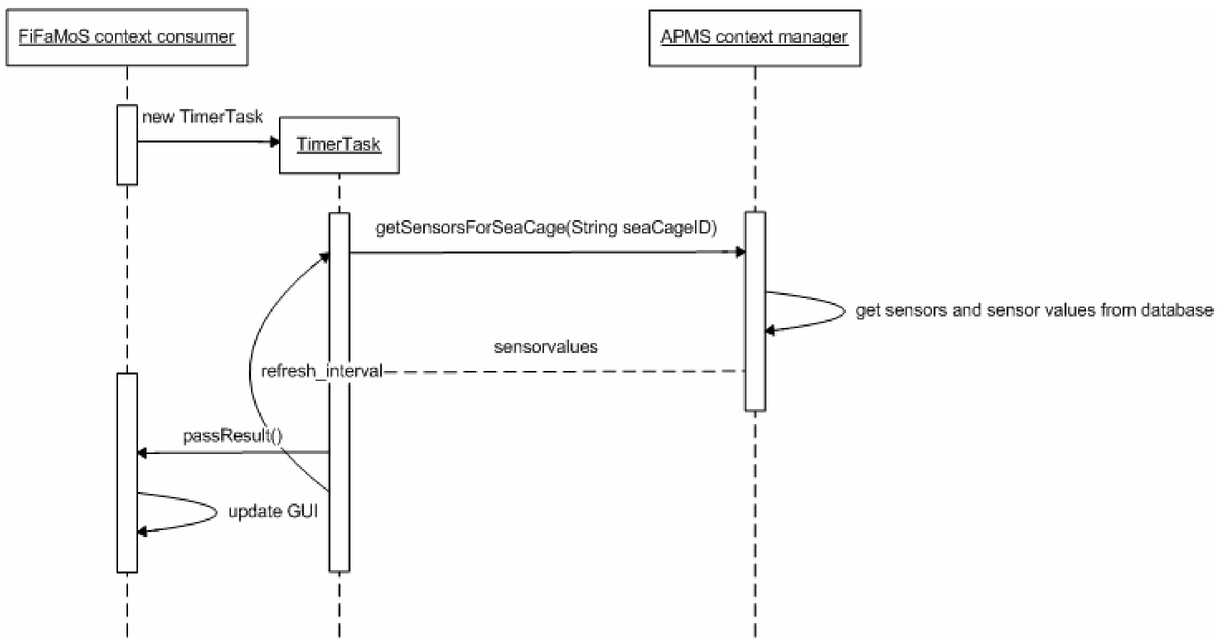


Figure 23: Get sensor info sequence chart

If a user chooses to view historical sensor values for a given period, the getHistoryForSensor method is invoked, and the context manager returns the sensor values with timestamps. The client will then draw a graph based on these values.

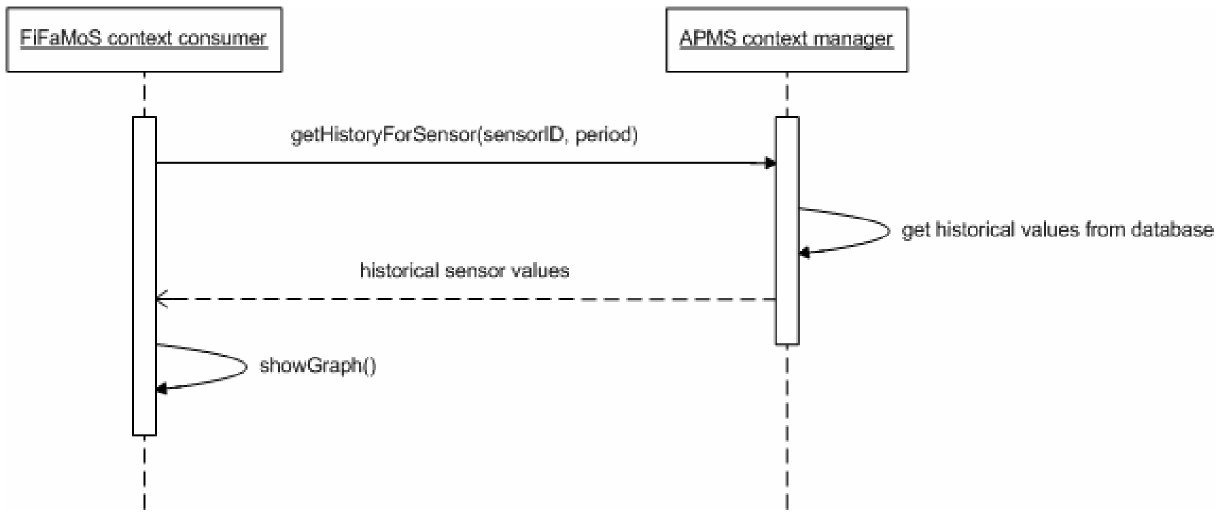


Figure 24: Get historical sensor values sequence chart

3.3.4.2 FiFaMoS context source – APMS context manager (E-IC-XMLRPC1.1)

Figure 25 shows the sequence of message passed during normal operation from the context source to the context manager. When the context source (M2M module) is started, it will contact the context manager and ask for its configuration. This will be returned as an XML string that is parsed in the module. When the module is correctly configured, it will periodically invoke the *postSourceXML* (String xml) method on the context manager. The XML elements can contain new sensor values, GPS position or other commands that the context manager can execute. If the elements contain new sensor values, the context manager will store these in persistent memory, check if the values are within allowed limits, and send an XML response to the M2M module. If some of the sensor values are out of bounds, an alarm message will be added to the XML response. The module will parse this, and send an alarm as SMS to the given fish farm operator (stated in the configuration). Also, if a new configuration is available, this will be noticed in the XML response as well.

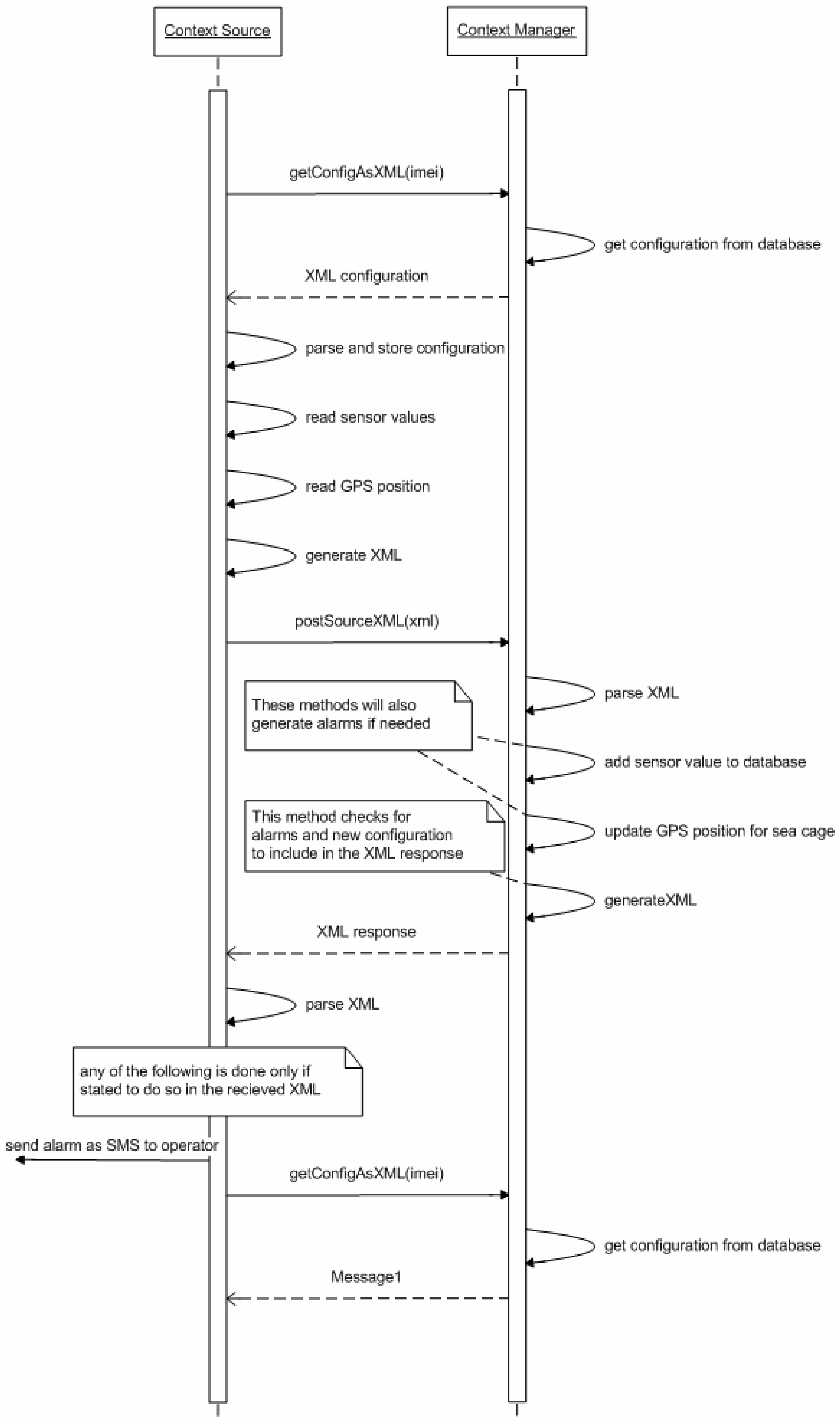

Context Source
Context Manager
getConfigAsXML(imei)
get configuration from database
XML configuration
parse and store configuration
read sensor values
read GPS position
generate XML
postSourceXML(xml)
parse XML
These methods will also generate alarms if needed
add sensor value to database
This method checks for alarms and new configuration to include in the XML response
update GPS position for sea cage
generateXML
XML response
parse XML
any of the following is done only if stated to do so in the recieved XML
send alarm as SMS to operator
getConfigAsXML(imei)
get configuration from database
Message1

Figure 25: Get configuration sequence chart

3.3.5 Activity diagrams

Activity diagrams represent the business and operational workflows of a system. An activity diagram shows the overall flow of control, including messages sent and received and processes that is run. There are many ways to represent the workflow with activity diagrams, and we have chosen to use the SDL notation.

3.3.5.1 Input component: main operational functionality

The main operational functionality of the context manager's input component is to receive XML from the context source, parse this and execute the commands provided in the XML. These are commands add (for sensor values) and update (for GPS position). When the sensor value is added and the position is updated, the input component also checks if the new value and position is within predefined limits. If this is not the case, an alarm is created and stored in the persitant memory. Finally an XML response is generated, and if there is any alarms that has not been sent, or if there's been a change in the configuration of the sea cage, it will be added to the XML response.

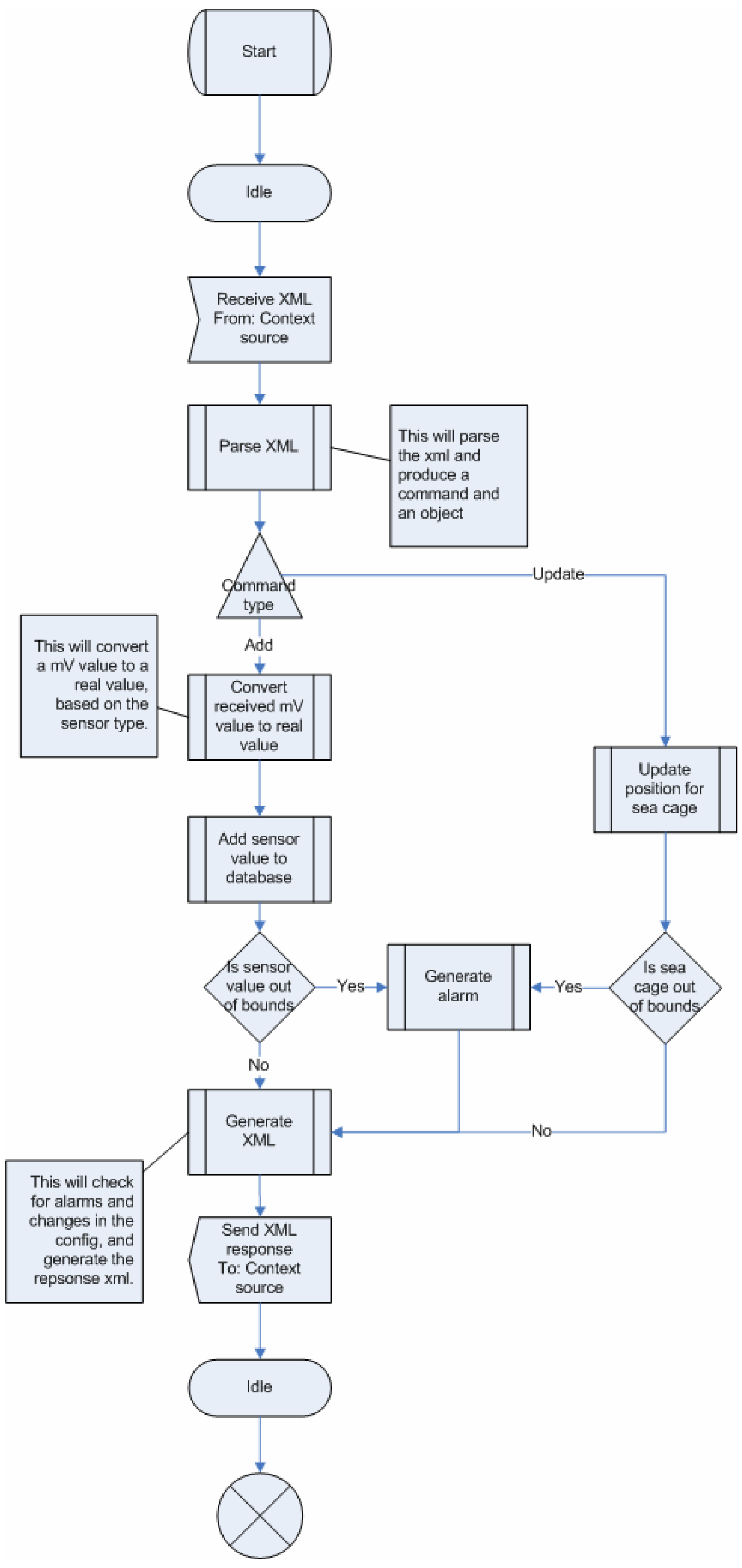

Start
Idle
Receive XML From: Context source
Parse XML
This will parse the xml and produce a command and an object
Command type
Update
Add
This will convert a mV value to a real value, based on the sensor type.
Convert received mV value to real value
Update position for sea cage
Add sensor value to database
Is sensor value out of bounds
Yes
Generate alarm
Yes
Is sea cage out of bounds
No
Generate XML
No
This will check for alarms and changes in the config, and generate the repsonse xml.
Send XML response To: Context source
Idle

Figure 26: Activity diagram showing the input component's main operational functionality

3.3.5.2 Input component: main administrative functionality

The main administrative functionality of the input component is to let the context consumer add, update and remove the different parts of the fish farm. This could be to add a frame, update a sea cage and remove a sensor. When the XML with the commands to be performed is received, it is parsed and the commands are executed.

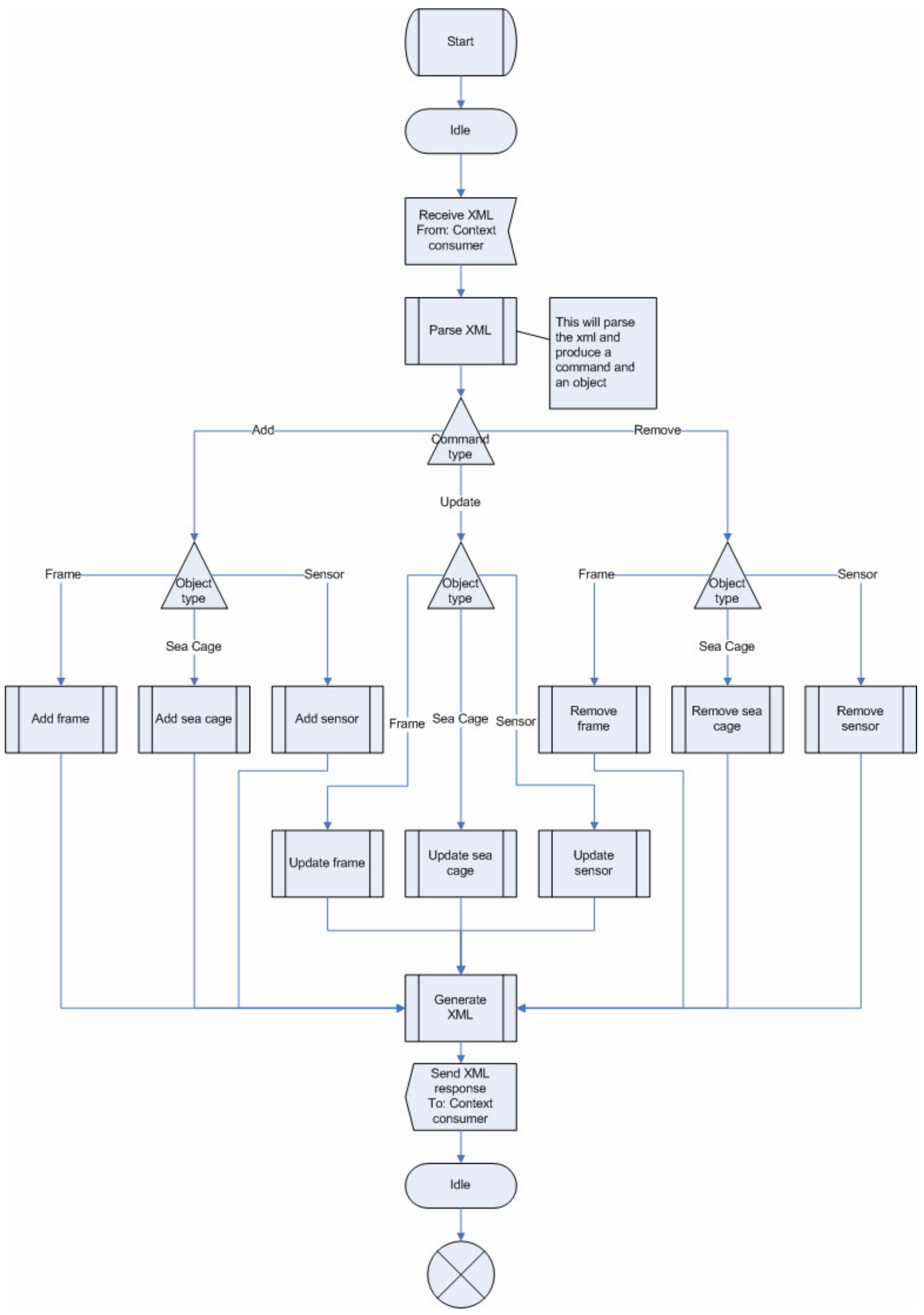


Figure 27: Activity diaram showing the input component's administrative functionality

3.3.5.3 Output component: Main operational functionality

The main operational functionality of the context manager's output component is the make the stored context (information about the fish farm) available to context consumers. The component waits for a request from the context consumer, and according to the request, the appropriate information is collected from the database and returned. The format of the returned result is XML for methods ending with AsXML, and CSV for the others.

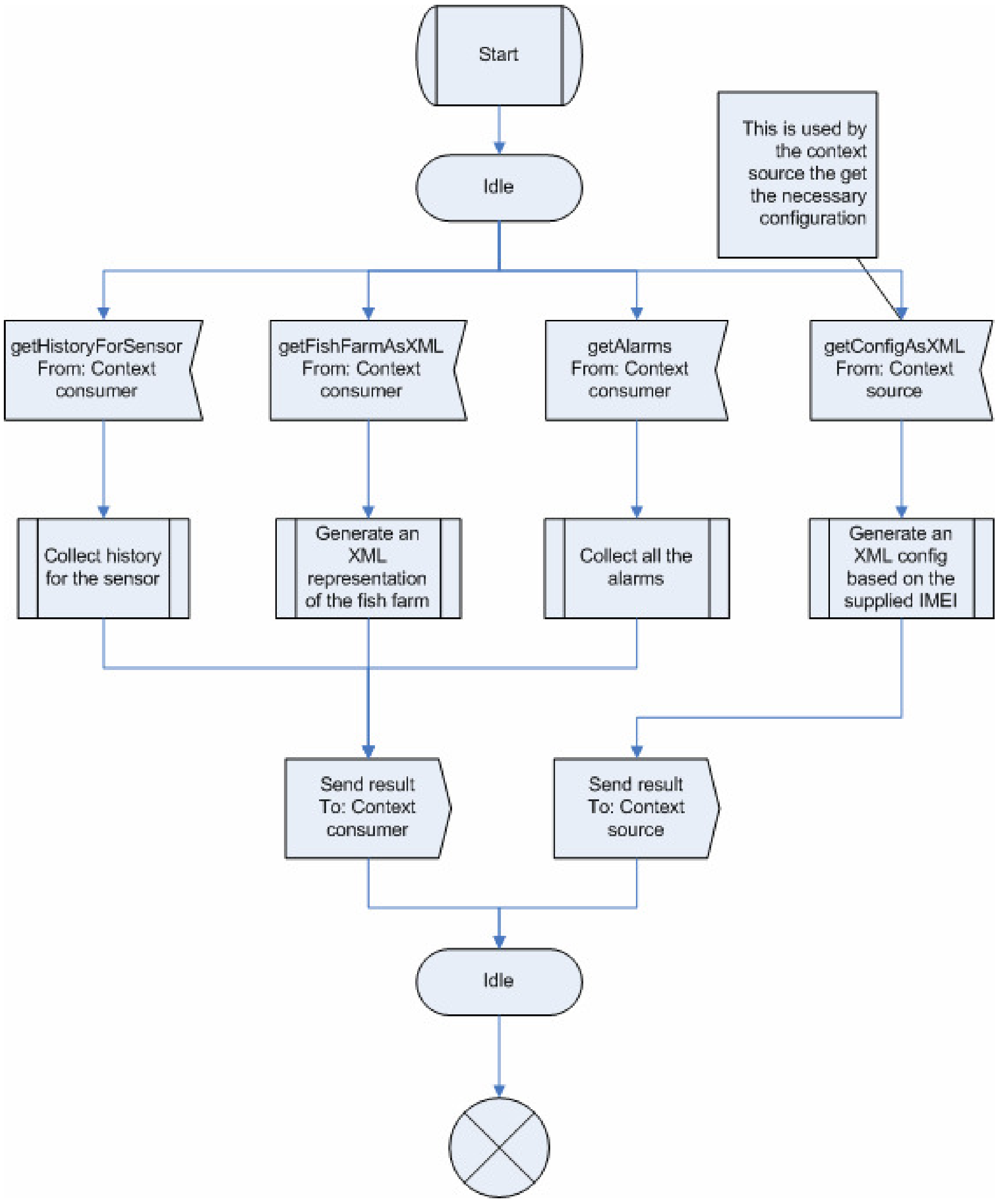

Start
Idle
This is used by the context source the get the necessary configuration
getHistoryForSensor From: Context consumer
getFishFarmAsXML From: Context consumer
getAlarms From: Context consumer
getConfigAsXML From: Context source
Collect history for the sensor
Generate an XML representation of the fish farm
Collect all the alarms
Generate an XML config based on the supplied IMEI
Send result To: Context consumer
Send result To: Context source
Idle

Figure 28: Acitvity diagram showing the output component's main operational functionality.

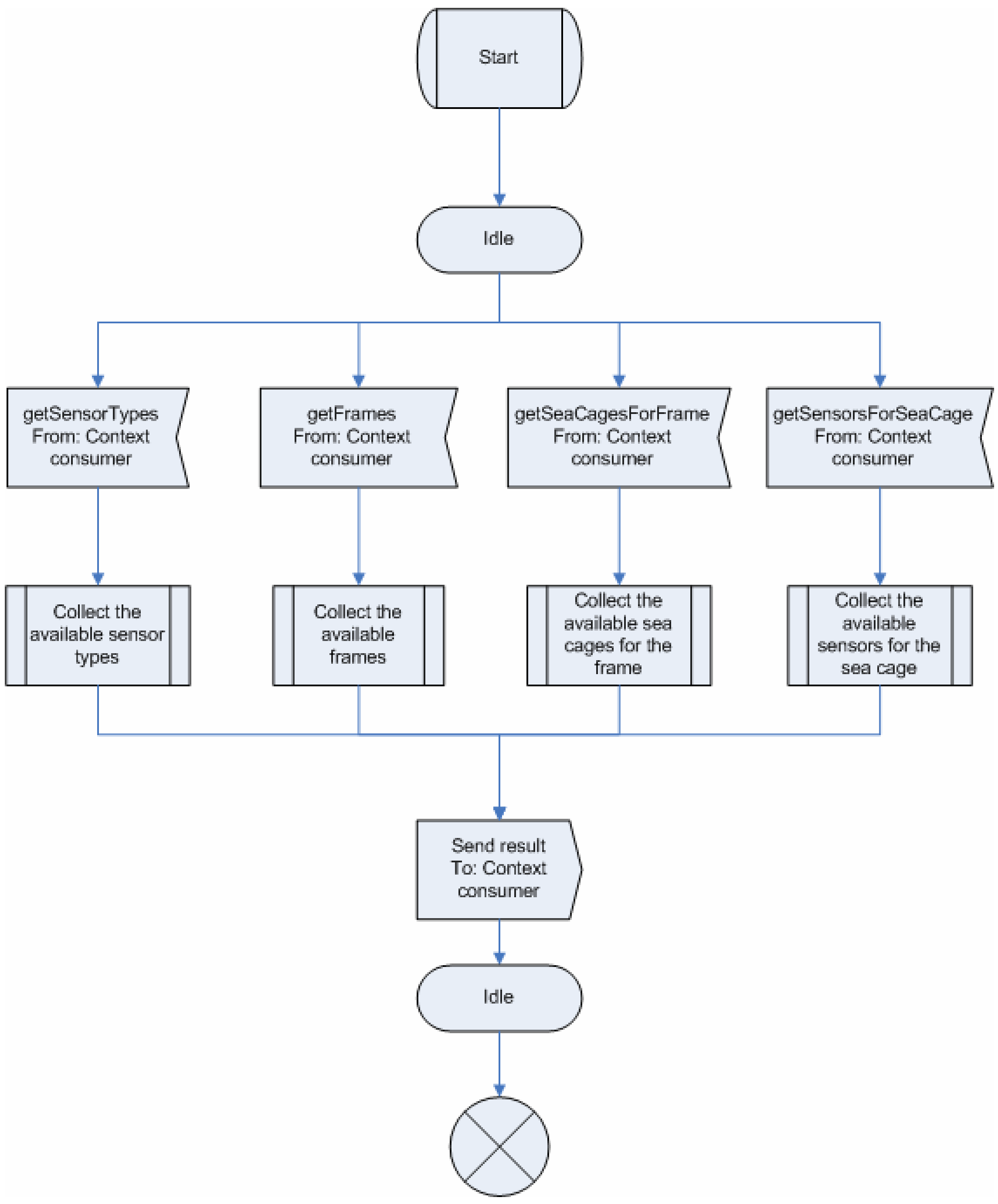

Start
Idle
getSensorTypes From: Context consumer
getFrames From: Context consumer
getSeaCagesForFrame From: Context consumer
getSensorsForSeaCage From: Context consumer
Collect the available sensor types
Collect the available frames
Collect the available sea cages for the frame
Collect the available sensors for the sea cage
Send result To: Context consumer
Idle

Figure 29: Output component's main operational functionality continued

3.3.5.4 Context source functionality

The context source starts with getting its own IMEI number for identification. Then it gets its configuration as XML (which is generated based on the IMEI number) from the context manager. When the configuration is parsed and stored, the context source starts to read sensor values and GPS position, wraps it in XML (with commands) and sends it to the context manager. As mentioned earlier, the context manager executes the commands and generates the respons XML. When the response XML is received in the context source, it is parsed and if there are any alarms, they are sent as SMS to the receiver stated in the configuraten. If there's a new configuration available, the context source asks for it.

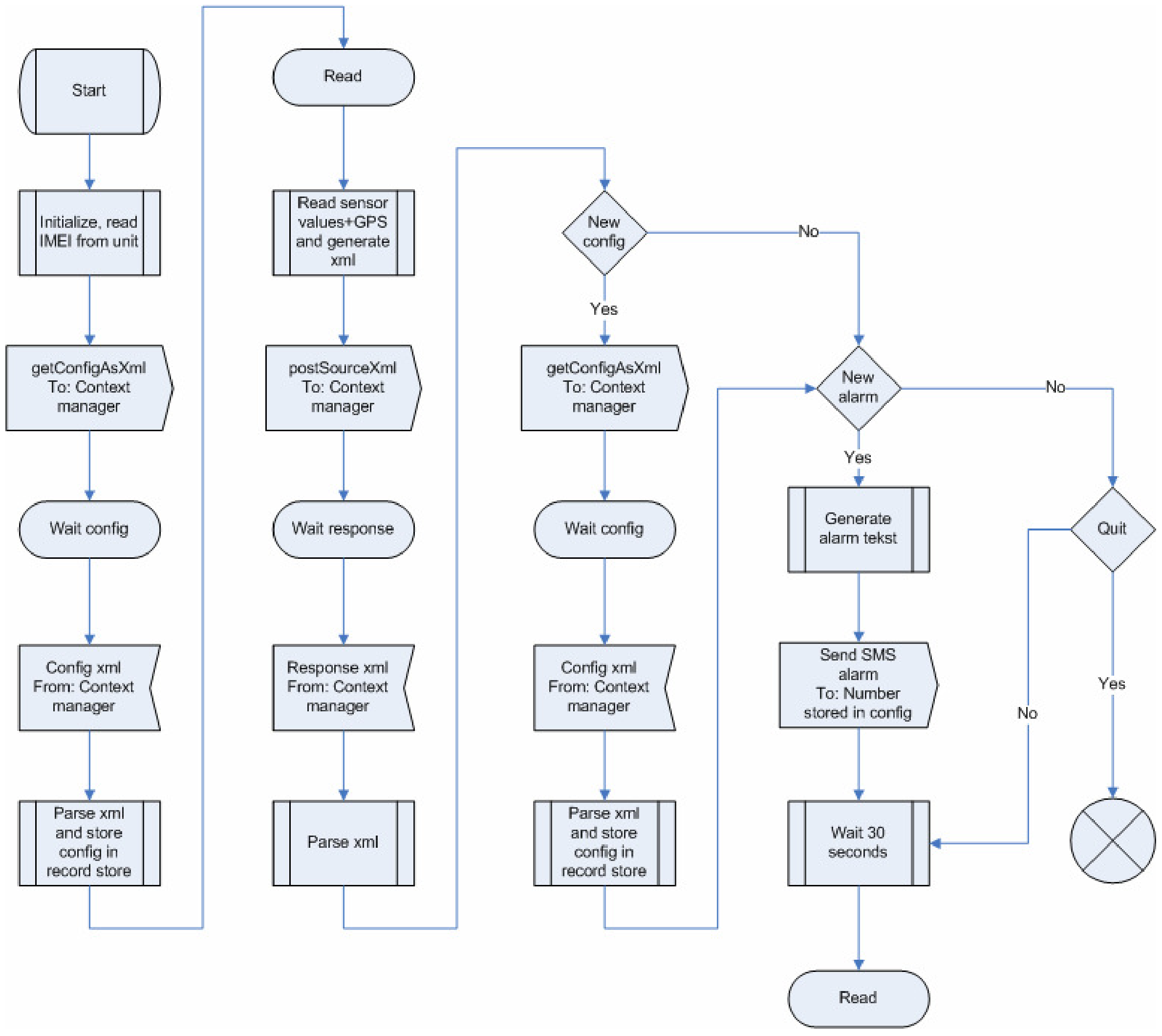

Start
Initialize, read IMEI from unit
getConfigAsXml To: Context manager
Wait config
Config xml From: Context manager
Parse xml and store config in record store
Read
Read sensor values+GPS and generate xml
postSourceXml To: Context manager
Wait response
Response xml From: Context manager
Parse xml
New config
No
Yes
getConfigAsXml To: Context manager
Wait config
Config xml From: Context manager
Parse xml and store config in record store
New alarm
No
Yes
Generate alarm tekst
Send SMS alarm To: Number stored in config
Wait 30 seconds
Read
Quit
Yes
No

Figure 30: Acitvity diagram showing the context source functionality

3.3.5.5 FiFaMoS Context Consumer

3.3.5.5.1 Initialize context consumer

First, the client will contact the context manager to retrieve information about the fish farm. This information is about the frames and sea cages of the farm. Then the fish farms and sea cages will be placed in the navigation part of the GUI, and sensor values and graph for the first unit in the tree will be displayed. At last, the timer (see Figure 32) that makes the sensor information update itself must be started. The context consumer will then be in the *initialized* state.

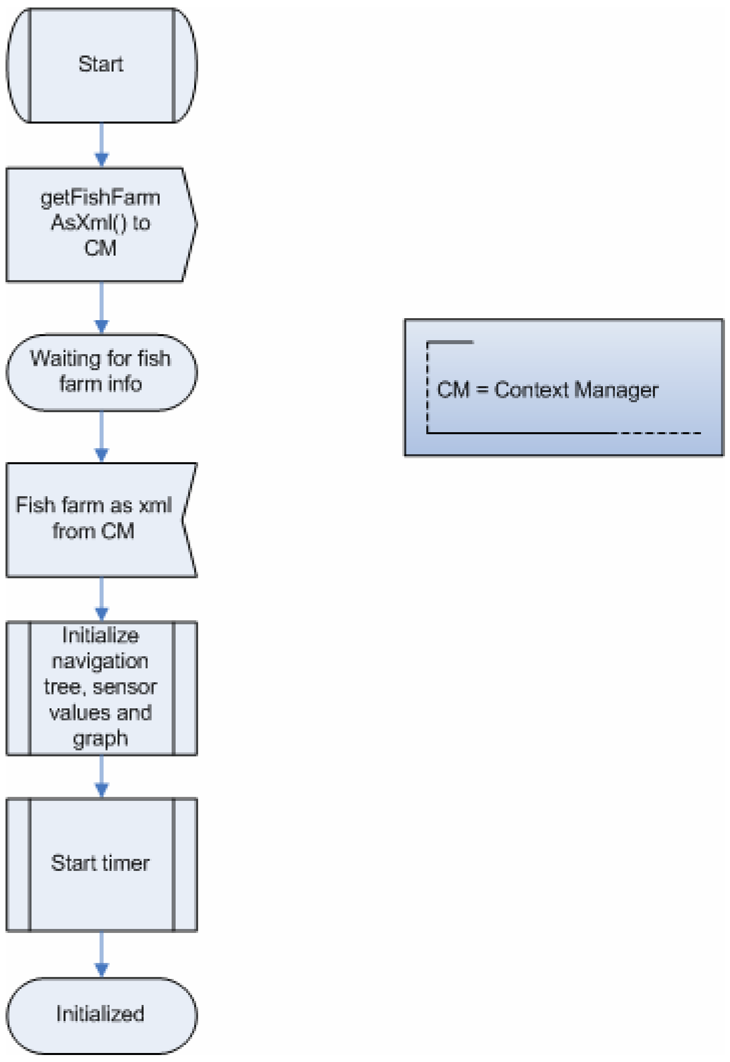

Start
getFishFarm AsXml() to CM
Waiting for fish farm info
Fish farm as xml from CM
Initialize navigation tree, sensor values and graph
Start timer
Initialized
CM = Context Manager

Figure 31: Activity diagram showing the initialization of the context consumer

3.3.5.5.2 *Context consumer operational functionality*

The operational functionality is functions that can be triggered from the main view in the program. The most important functions are to get current sensor information from a sea cage and to view graphs for a sensor. In addition, settings can be altered, or the program can be exited.

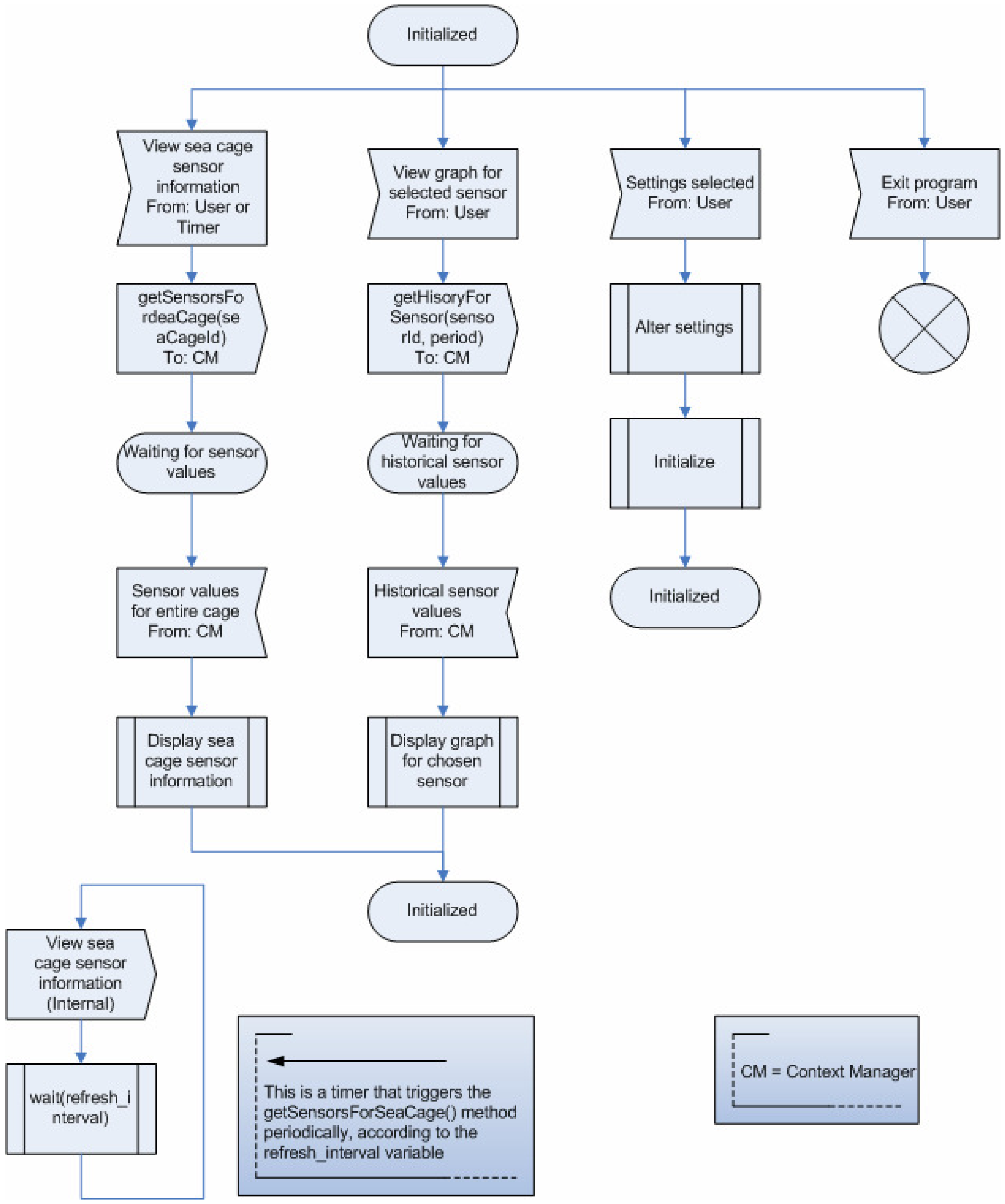

Initialized
View sea cage sensor information From: User or Timer
View graph for selected sensor From: User
Settings selected From: User
Exit program From: User
getSensorsFordeaCage(seaCageId) To: CM
getHisoryForSensor(sensorId, period) To: CM
Alter settings
Waiting for sensor values
Waiting for historical sensor values
Initialize
Sensor values for entire cage From: CM
Historical sensor values From: CM
Initialized
Display sea cage sensor information
Display graph for chosen sensor
Initialized
View sea cage sensor information (Internal)
wait(refresh_interval)
This is a timer that triggers the getSensorsForSeaCage() method periodically, according to the refresh_interval variable
CM = Context Manager

Figure 32: Activity diagram showing the operational functionality of the context consumer

3.4 Implementation

In this chapter, various implementation choices will be mentioned. The different modules and applications of the system will be discussed one by one, and at the end some encountered problems and problem solutions will be described.

3.4.1 APMS context manager components

In the following section a description of classes and their most important methods used in the context manager will be presented.

3.4.1.1 Input component

In this section the input component and its functions are at focus. As mentioned before, the input component is responsible for receiving data and storing it to persistent memory. In addition tasks like context processing and alarm triggering can be done in the input component.

3.4.1.1.1 Methods available via XML-RPC

The input component is described in Input component meta file (see appendix B.1), and the following methods are defined in the InputMBean-interface, and are made available to the web service:

- postConsumerXML(String)
    - This method is used for administrative operations from the FiFaMoS Context Consumer. See Code 4: Example of a context consumer XML.
- postSourceXML(String)
    - This method is used by the context source to post new context information, sensor values and GPS information. See Code 5: Example of a context source XML.
- confirmAlarms()
    - This method is used by the context consumer to confirm *all* alarms that has been generated
- fillDB(int, int)
    - This method is for testing purposes only. The first **int** is the number of frames and the second **int** is the number of sea cages per frame. The method creates 6

sensor types (oxygen, sourness, water throughput, light, temperature and food level), and a number of frames with a number of sea cages each. And finally adds one of each sensor to all the sea cages.

- cleanDB()
  - This method cleans the database. It deletes all entries in all tables.

3.4.1.1.2 Other methods in the input component

The following methods are included in the input component, but *not* available through web services (XML-RPC). These methods are for internal use only.

- init()
  - If anything is needed to be initialized before the component can be used, this is where to do it. The method is invoked by the context manager on start-up.
- addFrame(Frame)
  - This is a method for adding a frame to the system. The parameter is a Frame object based on the received XML.
- updateFrame(Frame)
  - This is a method for updating a frame. The parameter is a Frame object containing the new info
- removeFrame(int)
  - This method is for removing a frame. The parameter is an int, representing the id of the frame to be removed
- addSeaCage(SeaCage)
  - This method is for adding a sea cage to the system. The parameter is a SeaCage object based on the received XML
- updateSeaCage(SeaCage)
  - This is a method for updating a sea cage. The parameter is a SeaCage object containing the new info
- updateGpsForSeaCage(GPS)
  - This method is for updating the sea cage position. This method also checks if the sea cage is out of bounds (to far away from the initial position)
  - This method also sets the initial position of the sea cage if no position has been set yet

- removeSeaCage(int)
    - This is a method for removing a sea cage. The parameter is an int, representing the id of the sea cage to be removed
- addSensor(Sensor)
    - This is a method for adding a sensor to the system. The parameter is a Sensor object based on the received XML
- updateSensor(Sensor)
    - This method is for updating a sensor. The parameter is a Sensor object containing the new info
- removeSensor(int)
    - This is a method for removing a sensor. The parameter is an int, representing the id of the sensor to be removed
- addSensorValue(SensorValue)
    - This method is for adding a sensor value. The parameter is a SensorValue object based on the received XML. This method also converts the raw millivolt value to a real value, based on the sensor type. In addition it checks if the senor value is out of bounds (higher or lower than the predefined limits)
- addSensorType(SensorType)
    - This method is for adding a sensor type to the system. The parameter is a SensorType object based on the received XML.
- generateXML()
    - This method is used for generating XML for the return statement of postSourceXML (). It checks for alarms and updates and creates an XML response with the necessary information.

3.4.1.1.3 Detailed description of important methods

The most important methods are described in more detail here.

3.4.1.1.3.1 init()

This method initializes the XML parser with the following code lines:

```
private XMLReader parser;
```

```
...
parser = XMLReaderFactory.createXMLReader();
parser.setContentHandler(new XMLEventHandler(this));
```

Code 2: Snippet from the init method

As shown in the code snippet, three different classes are used; XMLReaderFactory, XMLReader and XMLEventHandler. The XMLReaderFactory is used only to create an instance of the XMLReader. They are both a part of the org.xml.sax package [50]. The parser needs a content handler to do something with the parsed XML. The method setContentHandler takes care of this. The XMLEventHandler is the implementation of the org.xml.sax.DefaultHandler class which is the class that does XML processing when different parts of the XML document are reached. The XMLEventHandler takes the XMLEventListener interface as a parameter in the constructor, and this is used to pass the processed result back to the input component.

```
...

parser.parse(
  new InputSource(
    new ByteArrayInputStream(
      xml.getBytes()))
  );
...
```

Code 3: Example of XML parsing

As shown in Code 3, the parse method takes an InputSource as an argument. The InputSource is a wrapper class for any source the XML document comes from, in this case a ByteArrayInputStream. The ByteArrayInputStream takes an array of bytes as an argument, so the getBytes() method is used on the XML string, *xml*.

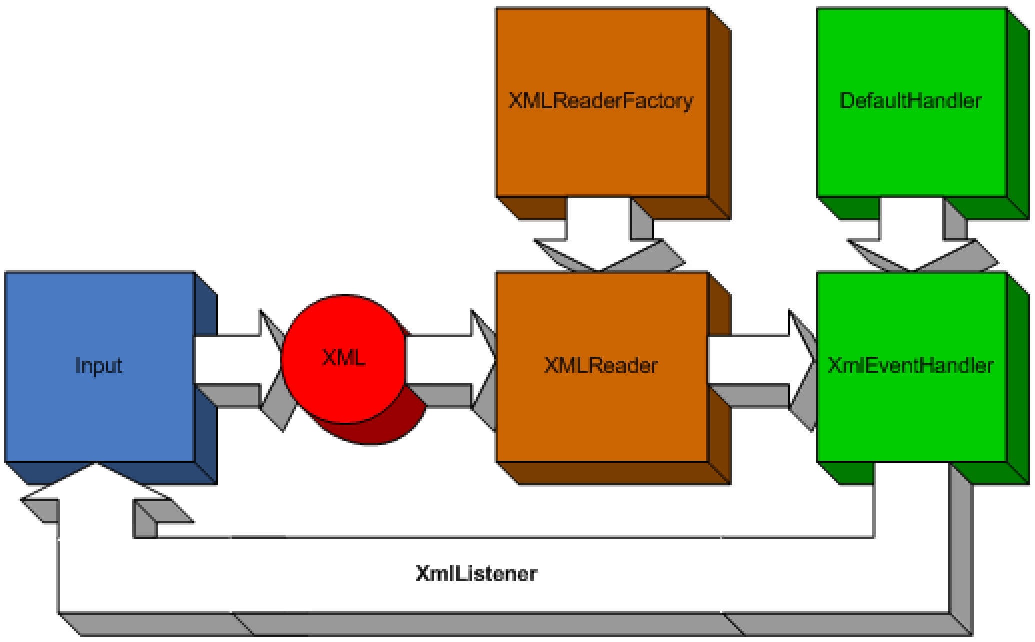


Figure 33: Illustration of how the XML parsing is implemented

When parsing, the Input class runs the XML through the XMLReader (which is the XML parser) using the parser.parse() method. All events that occur during the parsing are handled by the XMLEventHandler. This is events like startDocument, startElement, characters, endElement and endDocument (see Table 24). When an event is finished, the result is passed on back to the Input class via the XMLEventListener interface's method *passResult(Object)*.

Table 24: Different events generated during parsing of an XML document

Event	**Description**
startDocument	Occurs when start of document is reached (e.g. the document starts with <xml> but the startDocument is triggered before the <xml> tag is reached).
startElement	Occurs when start of an element is reached (e.g. <command>). This is a good place to initialize all that is needed during parsing of the element.
characters	Occurs when characters are reached (e.g. <description>Some

Event	Description
	description</description>). However, this event may occur several times on the same text. In this case one may have one event giving you the characters 'some des' and another giving 'cription'.
endElement	Occurs when the end of an element is reached (e.g. </command >). This is a good place to complete unfinished business regarding the element, and do whatever is needed (e.g. pass the result to the Input class),
endDocument	Occurs when the end of the document is reached (e.g. the document ends with </xml> but the endDocument is triggered after the </xml> tag is reached).

3.4.1.1.3.2 postConsumerXML()

The *postConsumerXML* method is used by context consumers for managing the system. That means add, remove and update different parts of the system, like frames, sea cages, sensor types and sensors.

```
<xml>
<command type="add">
<frame>
<id>0</id>
<location>Trondheimsfjorden</location >
<description>Testframe</description>
<mobile>99989796</mobile>
<email>operator@fifamos.net</email>
</frame>
</command>
</xml>
```

Code 4: Example of a context consumer XML

The root element is <xml>. It is necessary to have this root element, because the XML parser will set the first element as document start, and the document end will be after the end tag of that element. After document end is reached, no more parsing will be done.
The next element is <command> with the parameter type. This indicates that a command is to be executed. The parameter defines what type of command this is. In this case the type of command is 'add'. There are currently three different commands; add, update and remove. The add, update, and remove commands applies to frame, sea cage, sensor type and sensor.

The third element is <frame>. This is an XML representation of a frame. It contains the child elements <id>, <location>, <description>, <mobile> and <email>, which all add up to the properties of the frame. The <frame> element could just as well have been <seacage>, <sensor>, <sensortype> or <sensorvalue> (not used by context consumer), and the child elements would then have been changed according to the selected element.
When parsing the XML, some objects are created. First, a *Command* object is created. The command has two important fields; *String commandType* and *Object obj*. The commandType is one of the three mentioned above (add, update or remove) while the obj is an object representing the third element (here <frame>).

When the parsing is done, the Command object is sent back to the Input class via the XMLEventListener interface and executed there, according to the *commandType* and *obj* attributes.

3.4.1.1.3.3 postSourceXML()

The *postSourceXML* is used by the context source to post information on new sensor values and GPS position. The method also returns an XML document containing information about alarms and changes in the configuration of the system using the *generateXML()* method (see 3.4.1.1.3.5).

```
<xml>
<command type=”add”>
<sensorvalue>
<sensorid>1</sensorid>
<value>1375</value>
</sensorvalue>
</command>
<command type=”add”>
```

```
<sensorvalue>
<sensorid>2</sensorid>
<value>1187</value>
</sensorvalue>
</command>
<command type=”add”>
<sensorvalue>
<sensorid>3</sensorid>
<value>253</value>
</sensorvalue>
</command>
<command type=”update”>
<gps>
<seacageid>1</seacageid>
<latlng lat=”63 29 2” lng=”10 23 22” />
</gps>
</command>
</xml>
```

Code 5: Example of a context source XML

The example of source XML from Code 5 is very similar to the consumer XML in Code 4. The main difference is that only the add and update commands are used, and only sensor values are added and only GPS positions are updated.

In this example the object to add is <sensorvalue>, which is an XML representation of a sensor value. It contains the child elements <sensorid> and <value>, which makes up the sensor value. Further down a new command (update) with a <gps> element is reached. This is used for updating the position of the sea cage. The <gps> contains the elements <seacageid> and <latlng>. The latter is used for latitude and longitude coordinates.

Also when parsing this XML document, the Command objects and their appropriate objects are created. When the parsing is done, the Command object is sent back to the Input class via the XMLEventListener interface and executed according to commandType and obj.

3.4.1.1.3.4 addSensorValue()

This method performs one of the most essential tasks in the system, that is updating the system with a new sensor value. When a sensor value arrives as XML it is parsed and passed to this method. The *SensorValue* object here is *sv*. First of all the sensor for the sensor value is selected.

```
Select sel = new Select(”sensor”, getMetaData().getDbType());
sel.and(”id”, Select.EQUAL, sv.getSensorID());
List<Map<String, Object>> list = sel.execute();
Sensor sensor = new Sensor((HashMap) list.iterator().next());
```
Code 6: Code showing how to select the sensor for the sensor value

Then the sensor type for the sensor is selected. This is done for the conversion of millivolt to real values.

```
Select sel = new Select(”sensortype”, getMetaData().getDbType());
sel.and(”id”, Select.EQUAL, sensor.getSensorTypeId());
List<Map<String, Object>> list = sel.execute();
SensorType sensorType = new SensorType((HashMap) list.iterator().next());
```
Code 7: Code showing how to select the sensor type for the sensor

Now the actual conversion can be done.

```
int realSensorValue = Convert.convert(
sv.getValue(),
sensorType.getMaxRange(),
sensorType.getMinRange());
```
Code 8: Code showing how to convert from millivolt to real sensor value

When the conversion is done, it is checked whether the sensor value is out of bounds or not.

```
outofbounds = false;
if (realSensorValue > sensor.getMaxValue()||
realSensorValue < sensor.getMinValue())
outofbounds = true;
```
Code 9: Code showing how to check if the value is out of bounds or not

Finally the sensor value is added to the database.

```
Timestamp now = new Timestamp(new Date().getTime());
Insert in = new Insert(”sensorvalue”, getMetaData().getDbType());
in.column(”value”, realSensorValue);
in.column(”sensorid”, sv.getSensorID());
in.column(”received”, now);
in.execute();
```
Code 10: Code showing how to insert a sensor value into the database

If the sensor value *was* out of bounds, an alarm is created.

```
Insert in = new Insert(”alarm”, getMetaData().getDbType());
in.column(”seacageid”, s.getSensorID());
in.column(”sensorid”, sv.getSensorID());
in.column(”message”, message);
in.column(”issent”, 0);
in.column(”confirmed”, new Timestamp(0));
```

Code 11: Code showing how to add an alarm

3.4.1.1.3.5 generateXML()

Because all communication between the context source and the context manager has to be initiated from the context source, this method is used for generating an XML document as a reply to the web services call. The XML document generated may contain information about which alarms to send, and if a new configuration is available. The method may generate an XML document like this:

```
<xml>
<alarm id=”7”>
<seacageid>2</seacageid>
<sensorid>4</sensorid>
<message>The sea cage with ID 2 is 59m out of position!</message>
</alarm>
<updated frameid=”1” />
<xml>
```

Code 12: Example of XML response for the context source

Code 12 shows an alarm that has to be sent and a notice that tells the context source to update its configuration if it belongs to the frame with ID 1. All the necessary information about the alarm is included in the XML document, so if the context source had the sensor with the supplied ID (here: 4) all it has to do is to format and send the alarm as SMS to the number stored in the configuration.

There is no mail API in the context source used here (Aplicom N12i) so even though the e-mail address is supplied, the functionality to send alarms as e-mail is not implemented yet.

3.4.1.2 Output component

In this chapter the output component and its functions are at focus. As mentioned before, the main task of the output component is to send out requested data or context.

3.4.1.2.1 Methods available via XML-RPC

The output component is described in its meta file (see appendix B.2), and the following methods are made available in the web service interface:

- getFrames(String, String, String, String)
  - This is a method for getting all frames that matches a given search criteria. The parameters are location, description, mobile and e-mail.
- getSeaCagesForFrame(String)
  - This method returns all the sea cages for a given frame. The parameter is the ID of the frame in question.
- getSensorsForSeaCage(String)
  - This method returns all the sensors for a given sea cage. The parameter is the ID of the sea cage in question.
- getHistoryForSensor(String, String)
  - This is a method for getting historical values of a sensor. The parameter is the ID of the sensor in question, and the period in milliseconds.
- getConfigAsXML(String)
  - This method is used by the M2M module to get its configuration based on the current settings in the fish farm. The parameter is the M2M module's IMEI number.
- getFishFarmAsXML()
  - This method returns the entire fish farm configuration (set up) as an XML document.
- getGpsData(String)
  - This method is used to get the position of a sea cage. The parameter is the ID of the sea cage in question.
- getAlarms()
  - This is a method for getting all alarms that has *not* been confirmed.
- getSensorTypes()

- This method is used to get the available sensor types. It is used for administrative purposes. Adding a new sensor type gives the possibility to extend the functionality of the entire system.

3.4.1.2.2 Detailed description of important methods

The most important methods are described in more detail here.

3.4.1.2.2.1 getHistoryForSensor(String, String)

This method is used to get historical sensor values. The parameters are the ID of the sensor in question and the period of time one wish to study in seconds. Due to some problems with J2ME not handling long values, seconds had to be used instead of milliseconds (see section 3.4.7.5).

```
long now = new Date().getTime();
Timestamp t = new Timestamp(now - (Long.parseLong(period) * 1000));
Select sel = new Select(”sensorvalue”, getMetaData().getDbType());
sel.and(”sensorid”, Select.EQUAL, Integer.parseInt(sensorid));
sel.and(”received”, Select.GREATER, t);
sel.orderBy(”received”, Select.ASCENDING);
```
Code 13: Code showing how to select historical values from a sensor

Code 13 shows how to select historical values from a given sensor. By selecting all sensor values newer than the current time minus the period supplied, a range of sensor values are made available.

3.4.1.2.2.2 getFishFarmAsXML()

This method generates an XML document that represents the configuration of the fish farm, and is used by the context consumer. It is very convenient to use for speed and flexibility. The following code shows how to generate an XML document representing the structure of the fish farm:

```
StringBuffer xml = new StringBuffer();
xml.append("<xml>");

Select selFrame = new Select("frame", getMetaData().getDbType());
List<Map<String, Object>> listFrame = selFrame.execute();
Iterator itFrame = listFrame.iterator();
while (itFrame.hasNext()) {
  HashMap frame = (HashMap) itFrame.next();
  xml.append("\n\t<frame>\n");
```

```
    ...

  Select selSeaCage = new Select("seacage", getMetaData().getDbType());
  selSeaCage.and("frameid", Select.EQUAL, (Integer) frame.get("id"));
  List<Map<String, Object>> listSeaCage = selSeaCage.execute();
  if(listSeaCage.size()<=0)
   xml.append("\t\t<seacage />\n");

  Iterator itSeaCage = listSeaCage.iterator();
  while (itSeaCage.hasNext()) {
   HashMap seacage = (HashMap) itSeaCage.next();
   xml.append("\t\t<seacage>\n");

     ...

   Select selSensor = new Select("sensor", getMetaData().getDbType());
   selSensor.and("seacageid", Select.EQUAL, (Integer)seacage.get("id"));
   List<Map<String, Object>> listSensor = selSensor.execute();
   if(listSensor.size()<=0)
    xml.append("\t\t\t<sensor />\n");
   Iterator itSensor = listSensor.iterator();
   while(itSensor.hasNext()) {
    HashMap sensor = (HashMap)itSensor.next();
    Select selSensorType =
    new Select("sensortype", getMetaData().getDbType());
    selSensorType.and("id", Select.EQUAL,
    (Integer) sensor.get("sensortypeid"));
    List<Map<String, Object>> listSensorTypes = selSensorType.execute();
    Iterator itSensorTypes = listSensorTypes.iterator();
    String sensorType = "";
    if(itSensorTypes.hasNext()) {
     HashMap map = (HashMap) itSensorTypes.next();
     sensorType = (String) map.get("type");

    }

    xml.append("\t\t\t<sensor>\n");

    ...

    xml.append("\t\t\t</sensor>\n");
   }
   xml.append("\t\t</seacage>\n");
  }
  xml.append("\t</frame>\n");
}
xml.append("</xml>");
```
Code 14: Code extract of how to generate an XML representation of the fish farm

3.4.1.2.2.3 getConfigAsXML()

This method generates an XML document with the correct M2M module configuration, based on the current set up of the fish farm. The configuration may look like this;

```
<xml>
  <frame value= ”1” />
  <seacage value=”1” />
  <sensorid value=”1” />
  <sensorid value=”2” />
  <sensorid value=”3” />
  <interval value=”30” />
  <mobile value=”47744774” />
  <email value=”operator@fifamos.net” />
</xml>
```

Code 15: Example of XML configuration of the M2M module

This configuration tells the M2M module the following:

- that it belongs to the frame with ID 1 and the sea cage with ID 1
- that it has three sensors with ID 1, 2 and 3
- that it shall send the sensor values at a rate of once per 30 seconds
- that the cell phone number to send the alarms to is 47744774
- that the email to send the alarms to is operator@fifamos.net (not implemented yet)

If any changes are done to the configuration of the fish farm, a notice will be sent to the M2M module, and the M2M module will automatically update its configuration (see 3.4.1.1.3.5).

3.4.1.3 Other classes

In this part the different classes used in the context manager are described.

3.4.1.3.1 com.telenor.apms.fifamos.objects

Several classes are common for more than one component of the system. Both the context manager and the context consumer use the same objects when working with the representation of the fish farm. In this section these objects are described.

- Frame
    - This class is the representation of a frame. It has the information on who is the responsible for this frame (both cell phone number and e-mail), which is used by the system to send alarms. It is also the “parent node” of sea cage.
- SeaCage

- This class represents the sea cage. It holds information about which M2M module that belongs to it, which frame it belongs to, and the current position given by a GPS coordinate (longitude and latitude).

- SensorType
  - This is the representation of a sensor type. It contains information on which sensor type this is, which unit is uses, and its maximum and its minimum range.
- Sensor
  - This class represents the realization of a sensor type. It has a maximum and a minimum value, and it knows which sensor type it is and which sea cage it belongs to.
- SensorValue
  - Each sensor produces sensor values, and it's represented by this class. It knows which sensor it belongs to and the sensor value.
- Command
  - This class represents a command that is passed between the nodes in the system. A command holds the type of command (add, update, remove etc) and the object to perform the command on (frame, sea cage, sensor etc). The command can be generated anywhere, but it is executed on the context manager.
- GPS
  - This is a representation of the GPS information that is gathered at the context source. It contains information on number of satellites, altitude, latitude and longitude.

3.4.1.3.2 coms.tools.relational

This package contains components for the abstraction of SQL sentences. Only Insert, Select and Delete was already implemented.

- Update
  - Because the context manager didn't include an abstraction of the update functionality for the database, a component for updating an entry in the

database was made (see appendix B.6). It is based on the Insert class in the same package, but generates an update SQL statement instead of insert.

3.4.1.3.3 com.telenor.apms.fifamos.utils

Different utilities that were used in the context manager were gathered in this package. Below is a brief description of the different classes.

- XMLEventHandler
  - When a SAX parser is created, a separate class is used for handling events that occur during parsing. This is events like the start or end of a document or an element (see Table 24). In this case XMLEventHandler is used. To use the parsed result somewhere else, an interface is needed.
- XMLEventListener
  - This is an interface for communication between the XMLEventHandler and whatever class that needs the parsed result. In this case it is used to pass the parsed result to the Input class.
- Convert
  - Described in 3.4.1.3.3.1 below.

3.4.1.3.3.1 Convert

This class used by the context manager for converting millivolt values sent from the context source to real/correct values based on a value range. In this case the range is defined in the SensorType class.

```
public static double convert(int mV, int maxrange, int minrange) {
// factor
 double A = (maxrange - minrange)/MILLIVOLT_MAX; //MILLIVOLT_MAX = 2800

// adjustment
 double B = minrange;

 return A*mV+B;
}
```

Code 16: Code showing how to convert from a value read by the context source to a real value

As shown in Code 16, the correct value is converted based on the maximum range and the minimum range of the sensor type, together with the maximum input value from the Aplicom

N12i module. This means if the context source read 1400mV on a pH sensor, the method would be called by convert(1400, 0, 14). The math would then be
((14-0) / 2800 * 1400) + 0 = 7

3.4.1.3.4 uk.me.jstott.jcoord

In need of a way to handle GPS coordinates and to have a quick and easy way of converting between different kinds of GPS coordinates, an implementation by Jonathan Stott [51] is used. The class LatLng is the most interesting one.

3.4.1.3.4.1 LatLng

This class holds the information on the latitude and longitude, and provides functionality for converting between different coordinates and for measuring distance between two points (two [latitude, longitude] coordinates). The result of the measurement is in kilometres, so one must remember to adjust the result to what ever is wanted.
Measuring the distance is implemented like this;

```
public double distance(LatLng ll) {
  double er = 6366.707;

  double latFrom = Math.toRadians(getLat());
  double latTo = Math.toRadians(ll.getLat());
  double lngFrom = Math.toRadians(getLng());
  double lngTo = Math.toRadians(ll.getLng());

  double d =
    Math.acos(Math.sin(latFrom) * Math.sin(latTo) + Math.cos(latFrom)
    * Math.cos(latTo) * Math.cos(lngTo - lngFrom))
    * er;

  return d;
}
```

Code 17: Code showing how to measure the distance between to latitude/longitude coordinates

Conversion from [latitude, longitude] to UTM was intentionally meant for easier measurement of the distance between two points, but with this functionality already implemented in the LatLng class, it isn’t necessary.

3.4.2 FiFaMoS Context Consumer

The client for personal computers is implemented in Java SE 5.0. It is divided into three packages: *com.telenor.apms.fifamos.client.j2se.gui, com.telenor.apms.fifamo.client.j2se.utils* and *com.telenor.apms.fifamos.objects*. The client also uses the two Apache packets: *commons-codec-1.3.jar* and *xmlrpc-2.0.1.jar*. Class diagrams are shown in Appendix F.

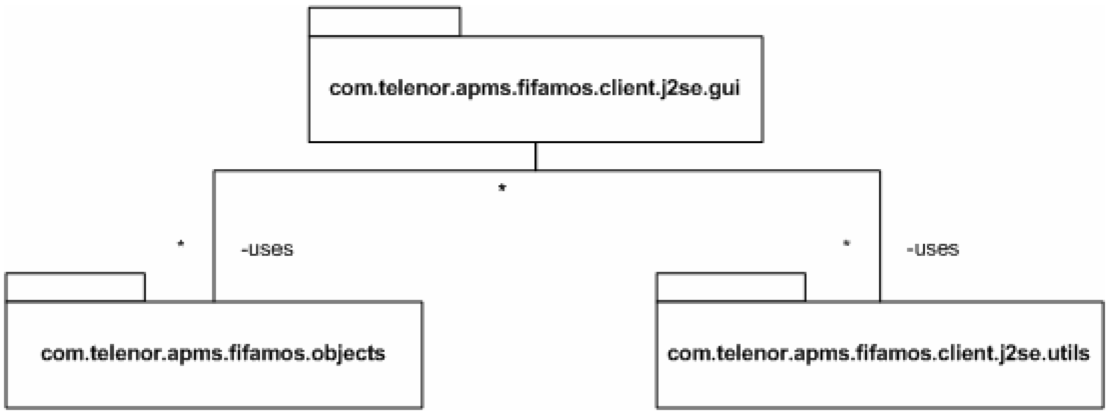

com.telenor.apms.fifamos.client.j2se.gui
*
*
-uses
*
-uses
com.telenor.apms.fifamos.objects
com.telenor.apms.fifamos.client.j2se.utils

Figure 34: FiFaMoS Context Consumer package overview

com.telenor.apms.fifamos.client.j2se.gui contains the main class (*FiFaMoS.java*), the settings dialog, and some input-boxes. These classes are implemented using the Java Standard Widget Toolkit, SWT. This will make the program have the same look and feel as other programs using the operating system's interface. The window is divided into four main parts: A list of fish frames and sea cages, a list of sensors and sensor values for the sea cage, a status text box and a graph showing historical sensor values. The list of fish frames and sea cages is implemented using a tree widget, where each tree item has its own fish frame or sea cage object. This makes navigation between the different fish farms and sea cages easy.

Once a sea cage is chosen in the tree, sensor information is displayed in the top right area of the window. Each sensor is displayed with the sensor type, current sensor value, and a button for showing historical sensor values in the graph. This list of sensor values is refreshed periodically using a *TimerTask*:

```
//schedule RPC-call for updating sensorinfo
si = new SensorInfo(String.valueOf(currentSeaCage.getId()), this);
timer = new Timer();
timer.scheduleAtFixedRate(new TimerTask() {
 public void run() {
  display.asyncExec(si);
 }
}, 50, Constants.REFRESH_INTERVAL);
```
Code 18: Code for refreshing the sensor values

Here, a *SensorInfo* thread is created periodically, according to the *Constants.REFRESH_INTERVAL* variable. This thread is executed by display.asyncExec(), and gets the latest sensor values from the context manager, and sends them back to the main class as a Vector in the *passResult()* method. The *passResult* method then updates the GUI with the new sensor information. The reason for using display.*asyncExec()* is a feature in SWT that prevents other threads than the UI thread (i.e. the thread that creates the new Display()) from accessing the GUI. Using *new Thread().start()* would make the Thread-object the parent of the thread (not the UI thread), but using *display.asyncExec()* the UI thread becomes the parent and may interact with the GUI.

A drawback with SWT is that it does not contain a widget for drawing graphs, so this had to be implemented manually. This was done using a canvas widget. A canvas is a blank area that can be drawn to using for example the *drawLine()* or *drawString()* methods, with X and Y coordinates as references. Before the graph could be drawn, the dataset had to be normalized. The normalize method takes a two dimensional integer table, with the sensor values and timestamps, as input, and returns a new two dimensional table with values in the graph size domain. First a scaling factor was found for the X and Y values:

```
//Find Y-value factor
factorY = (height-20)/(double)maxY;

//Find X-value factor
double factorX = (width-20)/(maxX-minX);
```

Code 19: Finding the scaling factor

New values were calculated and stored in the return table:

```
//Store new Y- and X-values in result table
for (int i = 0; i < values.length; i++) {
  values[i][1] = (values[i][1] - (long)minX);
  res[i][0] = (int)(values[i][0]*factorY);
  res[i][1] = (int)(values[i][1]*factorX+20);
}
```

Code 20: Calculating new values

When the dataset was normalized, a for-loop drew the graph line point by point, using the drawLine(x1, y1, x2, y2) method:

```
for(int i =0;i<(toBeDrawn.length-1);i++) {
  g.drawLine(toBeDrawn[i][1], turn(toBeDrawn[i][0]), toBeDrawn[i+1][1],
turn(toBeDrawn[i+1][0]));
}
```
Code 21: Drawing the graph

Sea cage positioning information is also available in this section. The position of the sea cage is represented in latitude and longitude degrees. In addition, it is possible to view the location in a map by pressing a button. The map dialog takes latitude and longitude as parameters, and contacts a web-site that render an image where the position is marked. This image is received in the *MapDialog* class, and displayed in the GUI.

The bottom right area of the graphical user interface contains a status text box. Here user actions, critical sensor values and error messages are logged.

The settings dialog lets the user adjust parameters like server IP-address and sensor value refresh interval. It is also possible to view and alter the different fish frames, sea cages and sensors. These can be deleted, updated or created. To make the main window update itself when the settings dialog is closed, a return statement is added:

```
 public boolean openSettings() {
   shell.pack();
   shell.open();
   shell.setActive();

   while (!shell.isDisposed())
     if (!shell.getDisplay().readAndDispatch()) shell.getDisplay().sleep();
          return true; //return true when finished
 }
```
Code 22: openSettings() method from the Settings dialog class

com.telenor.apms.fifamos.client.j2se.utils contains functionality for input and output, both network communication and disk access. All communication from the java client to the context manager is via XML-RPC. Several implementations for J2SE exist, but we have chosen the package from Apache. The package file is named xmlrpc-2.0.1.jar, and makes the user able to create a XML-RPC client object. By running the *execute(method name, parameters)* method, the client contacts the XML-RPC server and the response is returned.

```
XmlRpcClient client = new XmlRpcClient (Constants.SERVER_URL);
String response = (String)client.execute ("Output.getSeaCagesForFrame",
params);
```

Code 23: Executing a remote procedure call

As mentioned the *com.telenor.apms.client.j2se.utils* package also takes care of the disk access. This is used when writing log files and settings to disk. The log files are automatically stored to the /Logs directory, and the settings are stored as a property-file in the main program directory.

com.telenor.apms.fifamos.objects is the package containing the objects describing the semantics of the fish farm. Objects described includes: frame, sea cage, sensor and alarm among others (see appendix F.1).

3.4.3 FiFaMoS Mobile Context Consumer

The client for mobile phones and smart phones (FiFaMoS Mobile Context Consumer) is implemented in Java J2ME, MIDP 2.0, and consists of tree packages: *com.telenor.apms.fifamos.client.j2me.utils*, *com.telenor.apms.fifamos.client.j2me.gui* and *com.telenor.apms.objects*. It also uses some tools from the Comtor J2ME package, and the XML-RPC client in kxml-RPC.

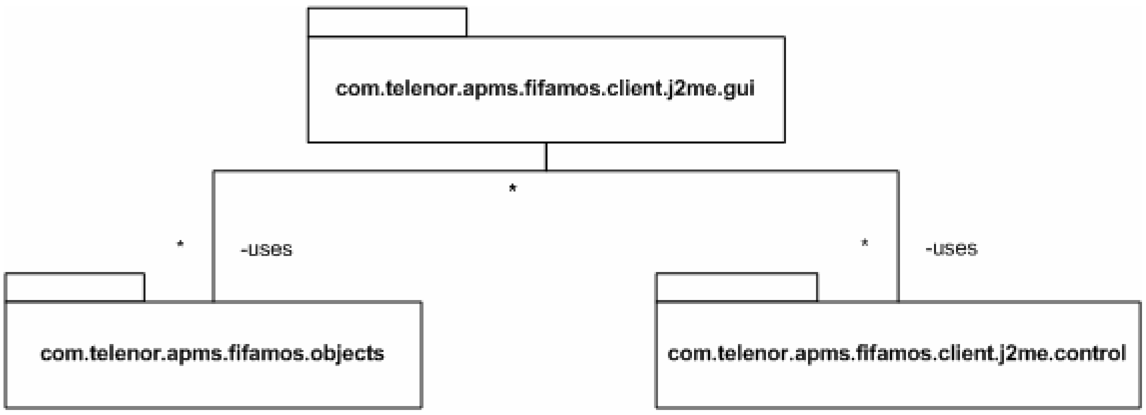

com.telenor.apms.fifamos.client.j2me.gui
*
*
-uses
*
-uses
com.telenor.apms.fifamos.objects
com.telenor.apms.fifamos.client.j2me.control

Figure 35: FiFaMoS Mobile Context Consumer package overview

The *com.telenor.apms.fifamos.client.j2me.control* includes the main *FiFaMoSMob* class as well as a RPC-handler class taking care of the communication. The application is based on several lists that the user can navigate through, and the purpose of the *FiFaMoSMob* class is to be a listener for these lists, and decide what is going to happen on user actions. After a user action, a new list will be displayed, and the RPC-handler class will be contacted to get data for the list. One special list is the list of sensors with current sensor values. This list is going to be updated regularly according to a user set interval. This refresh interval is set in the settings dialog, and stored in the phone's record store. To make the list update itself, a timer task is created in the list class:

```
   timer = new Timer();
   timer.scheduleAtFixedRate(new TimerTask() {
    public void run() {
     getSensorData();
    }
   }, 500, interval );
```

Code 24: Scheduling the getSensorData() method

The *getSensorData()* method contacts the RPC-handler, and updates the list with new data.

The RPC-handler is generic, and used by all lists (fish frame list, sea cage list, sensor list). To get data from the web service, you can just create an RpcHandler object, with method name and method parameters. The result will be passed back to the class via the *passResult* method.

```
public RpcHandler(String request, Vector params, RpcHandlerResult listener)
```

Code 25: The RpcHandler constructor

An XML-parser is not included in the application, so XML encoding of the data is currently not used. Instead, the values sent as a plain string where sensor values and other attributes are separated using either ":" or ";". To create the fish frame, sea cage or sensor objects, the resulting string is sent to the object's constructor, where a StringVector separated the attributes and creates the object. The StringVector class is from the Comtor J2ME utils package. This package also includes tools for formatting timestamps, which is used when drawing graphs.

Only one window in the FiFaMoS Mobile Context Consumer does not use lists, and that is the graph showing historical sensor values. This is implemented using the canvas from the low-level GUI API. Just like the java client, graphs are drawn using the *drawLine()* method with X and Y coordinates. The graph will adapt to the screen size, so that it will be easy to read on different phones.

The FiFaMoS Mobile application also has a settings window. This window can be accessed from the start menu, from the “settings” item in the list. The settings functionality on the FiFaMoS Mobile Context Consumer can not be used to setup the fish farm, but it is used to assign the correct server address, and to set the refresh interval of the sensor values. The latter parameter is important in the FiFaMoS Mobile Context Consumer, as this will affect the amount of GPRS traffic generated.

3.4.4 FiFaMoS context source application

The context source application is an IMlet for the M2M module is written in Java 2 Micro Edition (J2ME). IMlet [44] is a J2ME application that runs on the Information Module Profile (IMP) environment. IMP is a strict subset of the Mobile Information Device Profile (MIDP) commonly used in mobile phones with the distinction that it does not have a user interface (UI). This part of the system is used to supply the context manager with context information like sensor values. The package organization is shown in Figure 36, and the functionality is further described afterwards.

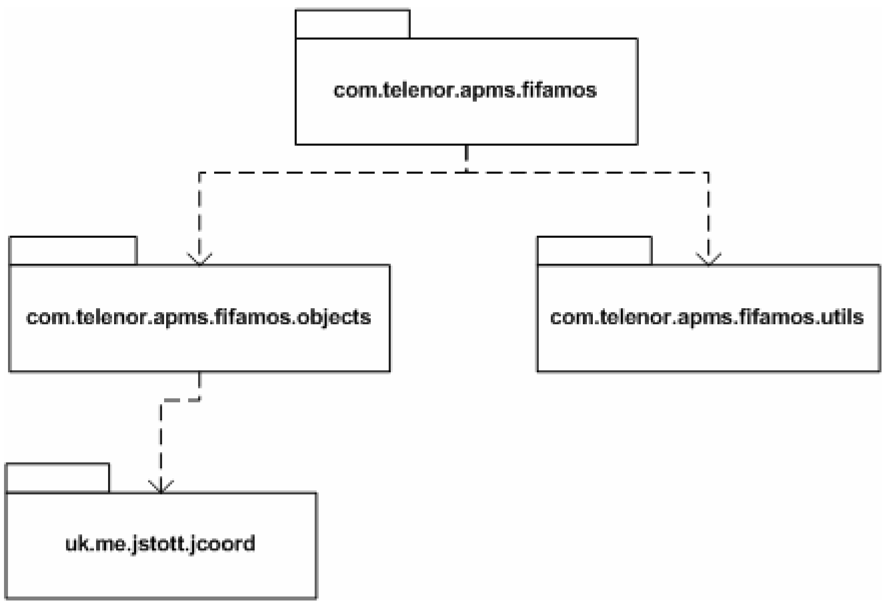

com.telenor.apms.fifamos
com.telenor.apms.fifamos.objects
com.telenor.apms.fifamos.utils
uk.me.jstott.jcoord

Figure 36: FiFaMoS Context Source package diagram

3.4.4.1 Initialization

On start up, the application reads the IMEI number from the M2M module. This is used to identify the module, and to determine which sea cage the M2M module is situated on. The application calls the method getConfigAsXML from the context manager, with the IMEI as parameter, then parses and stores the received configuration.

```
// Resolve the Embedded Terminal object
String etUrl =  ”corbaloc::127.0.0.1:19740/ORB/OA/IDL:ET:1.0”;
org.omg.CORBA.Object etRef = orb.string_to_object(etUrl);
ET et = ETHelper.narrow(etRef);
imei = et.IMEI();
Vector params = new Vector();
params.addElement(imei);

String xmlconfig =
(String) xmlrpc.execute(”Output.getConfigAsXML”, params);
Config.saveConfig(xmlconfig);
```

Code 26: Code showing how to retrieve the IMEI number, and to initialize the M2M module

3.4.4.2 Reading and sending the sensor values

When the context source is in normal operation, it periodically reads the sensor values as millivolt values from its analogue inputs or as high and low from its digital inputs as shown in Code 27.

```
IOControl ioc = IOControl.getInstance();
...
int aValue = ioc.getAnalogInputPin(ANALOG_INPUT_PIN);
boolean dValue = ioc.getDigitalInputPin(DIGITAL_INPUT_PIN);
```

Code 27: Code showing how to read from input pins

These values are wrapped in XML (see Code 5) and sent to the context manager via XML-RPC over the GPRS/EDGE network”.

```
Vector params = new Vector();
StringBuffer xml = new StringBuffer();
xml.append(“<xml>”);
//XML for sensor values and GPS data
...
xml.append(“</xml>”);
params.addElement(xml.toString());
String result = (String) xmlrpc.execute(”Input.postSourceXml”, params);
```

Code 28: Code showing how to send context to the context manager

The response from a sensor value post (a context message sent) is a new XML document describing the next actions for the context source (see Code 12). If an alarm is to be sent, or new configuration is available, this will be notified here.

As mentioned, the context source is using kXML-RPC [20] for communication with the context manager. The commands are sent as shown in Code 23. This is done to achieve a connectionless service oriented architecture (SOA) [49] and to reduce traffic cost.

When the sensor values are received in the context manager, they are marked with a timestamp. This is done to keep track of when the value where received, for historical purposes.

3.4.4.3 GPS functionality

To keep track of where the sea cage is at all times, a GPS receiver is connected to it and the position is read periodically. For easier use of the GPS functionality the Aplicom N12i provides, a wrapper class was created. The class *GPSControl* (in the package *com.telenor.apms.fifamos.n12.utils*) is the wrapper for Aplicom N12i's GpsModule class, used to read NMEA (see 2.2.7.1) from a GPS receiver connected to a serial port. The main purpose of the GPSControl is to read the NMEA data, extract the relevant information and make it available to others. This is done in a separate class (see appendix B.5) for less complexity in the program, and is very useful for later reuse of the code.

```
Hashtable props = new Hashtable();
props.put("com.nokia.m2m.orb.UseM2MGateway", "no");
org.omg.CORBA.ORB orb = ORB.init(null, props);
...
GPSControl gps = new GPSControl(orb);
gps.read();

String latitudeDegrees = gps.getLatdeg();
... = gps.getLatmin();
... = gps.getLatsec();
// and gps.getLngdeg() etc for longitude
```
Code 29: Code showing how to use the GPSControl class

As shown in Code 29, the usage of the *GPSControl* is quite simple. Just notice that *gps.read()* has to be called before any of the get-methods.

3.4.4.4 SMS functionality

To make sure alarms arrive in time for actions to be taken, they are sent by SMS to the person responsible for the sea cage in question. As with the GPS wrapper, an SMS wrapper also was made to make it easier to use the SMS functionality in the Aplicom N12i module. The class SMSControl is the wrapper class for the SMS functionality.

```
Hashtable props = new Hashtable();
props.put("com.nokia.m2m.orb.UseM2MGateway", "no");
org.omg.CORBA.ORB orb = ORB.init(null, props);
...
SMSControl sms = new SMSControl(orb);
sms.sendTextSMS(phoneNumber, message);
```
Code 30: Code showing how to use the SMSControl class

The Aplicom N12i module is used as an SMS gateway, so any alarms that are to be sent must be piggybacked from the context manager as a response to a *postSourceXML()* call (see Code 12 for an example).

3.4.4.5 Watchdog functionality

To ensure maximum up-time on the module, the watchdog functionality is used. This is a function that makes the module reset itself if a preset timer reaches zero. The timer is supposed to be reset to the initial value in given intervals. This is done in important loops in the application. If this loop stops (the module crashes), the watchdog timer will reach zero, and the module will be reset automatically. If the error is not a serious hardware fault, the application will work as normal after the reset. In this case the watchdog is reset before the sensor reading. This will ensure that the timer is reset at regular intervals, and that if the module stops sending sensor values (which is its main purpose), it will be reset automatically.

```
//initiate the watchdog
WatchdogTimer wdt = new WatchdogTimer();
wdt.setTimeout(5*60);
...
while(running) {
  //reset the timer at regular intervals, shorter than the timeout.
  wdt.resetTimer();
  ...
  /** read and send sensor values */
  Thread.sleep(interval);
}
```
Code 31: Code showing how to use the WatchdogTimer

3.4.4.6 Debug information over serial interface

When developing applications for the M2M module, a need for debugging appeared. For this a serial port logger implementation based on the SerialPortLogger from the IMlet programming guide [44] was used. This is a wrapper class for the serial port interface. It provides simplified methods for sending debugging information over the serial port interface.

```
SerialPortLogger.getInstance().write(message);
```
Code 32: Code showing how to use the SerialPortLogger

As shown in Code 32, its usage is very simple, but the functionality behind is a little more complex. This is shown in Code 33.

```
...
sc = (StreamConnection)
  Connector.open("comm:3;baudrate=115200", Connector.READ_WRITE);

// Open outputStstream
outStream = sc.openOutputStream();
...
if (outStream != null && msg != null) {

// prepare timestamp
Calendar cal = Calendar.getInstance();
// Print timestamp and message
msg = "<[" + cal + "] " + msg + ">\n";
outStream.write(msg.getBytes());
```
Code 33: Code showing a snippet of the implementation of the serial port logger

3.4.5 Testing

To verify that the sensor values were properly stored, and that the client was able to present the data, various tests where performed on the system. Since we were following the RUP model using two iterations, we had two major test periods in the process. The tests are described later in this chapter.

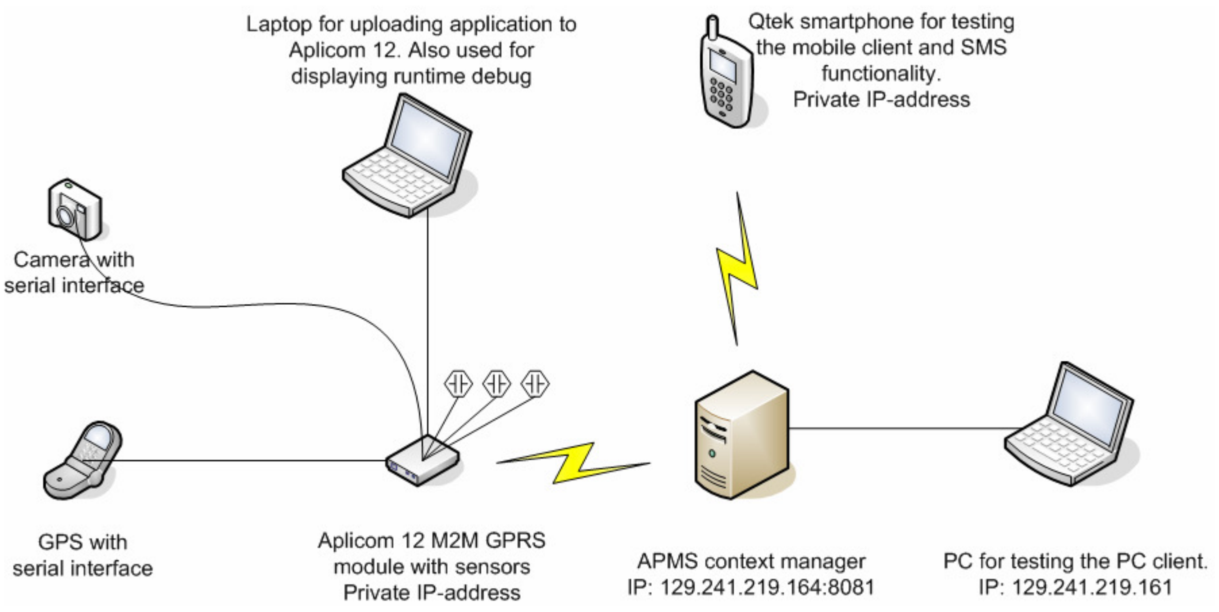

Laptop for uploading application to Aplicom 12. Also used for displaying runtime debug
Qtek smartphone for testing the mobile client and SMS functionality. Private IP-address
Camera with serial interface
GPS with serial interface
Aplicom 12 M2M GPRS module with sensors Private IP-address
APMS context manager IP: 129.241.219.164:8081
PC for testing the PC client. IP: 129.241.219.161

Figure 37: Testing environment

The testing environment were mostly as Figure 37 describes, but to save GPRS expenses and to ease operation, the Aplicom N12i M2M module was sometimes replaced with a java application simulating the module. This simulator (see appendix B.7) takes a table of sensor IDs and their respective minimum and maximum value, and fills the database with random values within the min-max-range. The simulator creates an xml document with one value for each sensor and uses XML-RPC to send it to the context manager. The context manager parses it and stores the values in the database.

There was also other times when we decided to use software solutions simulating the real-world hardware that was supposed to be used. That includes using the J2ME Mobile simulator from Wireless Toolkit 2.1. During implementation, eclipse was set up to run the application in this emulator. This way we did not have to pack the application and transfer it to the mobile phone. GPRS expenses were also saved.

The same method was used during the development of the software on the Aplicom N12i module. Using the Aplicom N12i concept simulator, IMlets could be tested without being transferred to the module.

3.4.5.1 Testing details

To verify that requirements FR1.* were fulfilled, dummy sensors (see appendix D.1) were connected to the M2M module. When adjusting the dummy sensors, the values could be seen both in the FiFaMoS Context Consumer and the FiFaMoS Mobile Context Consumer. After iteration one, all these requirements were fulfilled except FR1.1 which was planned implemented later in the project. Requirement FR2 was also working, which indicated that sensor values were stored in the database correctly.

The main tasks of iteration two was to add GPS functionality and to send alarms if sensor values were out of bounds. To test these requirements, the fish farm was configured with sensor value limits, and the dummy sensors were adjusted to trigger alarms. By the end of iteration two, this functionality worked. Alarms were displayed in the FiFaMoS Context Consumer, and sent to the mobile via SMS.

FR3.1 was on of the most important requirements to fulfil. This functionality was not easy to verify, as the GPS was connected to the M2M module via cable inside a house, where the GPS satellites were unavailable. To get this tested, we played a formerly recorded trace on the GPS receiver. The trace simulated movement in a triangle located in the Trondheim fjord. When the M2M module booted, it stored its initial position in the context manager, and continually sent its new position. If the difference between the two positions was higher than allowed, an alarm was triggered. By the end of iteration two, it worked, and the requirement was fulfilled.

Requirement FR5 describes the video surveillance functionality, and should after the plan be working after iteration two. This was not as easy as first anticipated, and the M2M module was unable to get image data from the camera. The test failed, and requirement FR5 was added to future work (see section 5.1 Future work).

All the other requirements passed the testing procedures.

Table 25: Test results

Requirement	Description	Test passed	Phase
FR1	Read sea cage sensor values	V	1
FR1.1	Read sea cage position (GPS)	V	2
FR1.2	Read sea cage net sensor	V	1
FR1.3	Read wave sensor	V	1
FR1.4	Read water level sensor	V	1
FR1.5	Read temperature sensor	V	1
FR1.6	Read water quality sensor	V	1
FR1.7	Read food level sensor	V	1
FR1.8	Read wind sensor	V	1
FR1.9	Read current sensor	V	1
FR1.10	Read pH sensor	V	1
FR1.11	Read oxygen content	V	1
FR2	Get historical sensor values	V	1
FR3	Send alarm	V	2
FR3.1	Send alarm when sensor values are out of range	V	2
FR3.2	Send alarm when a sea cage is out of position	V	2
FR3.3	Send alarm via SMS	V	2
FR4	Display alarm in application	V	2
FR5	Start surveillance camera session	X	2
FR6	Store logging data to file	V	2
FR7	Alter fish farm configuration from the PC client	V	2
FR7.1	Alter frame configuration	V	2
FR7.1.1	Add new frame	V	2
FR7.1.2	Update existing frame	V	2
FR7.1.3	Delete existing frame	V	2
FR7.2	Alter sea cage configuration	V	2

Requirement	Description	Test passed	Phase
FR7.2.1	Add new sea cage	V	2
FR7.2.2	Update existing sea cage	V	2
FR7.2.3	Delete existing sea cage	V	2
FR7.3	Alter sensor configuration	V	2
FR7.3.1	Add new sensor	V	2
FR7.3.2	Update existing sensor	V	2
FR7.3.3	Delete existing sensor	V	2

3.4.5.2 Measurements

To verify the requirements stated in chapter 3.2, some measurements were done. First we measured the storage needs, then the response times of the system. To be able to get a veritable result, the sensor simulator (see appendix B.7) was used to fill the database with the wanted data (frames, sea cages, sensors and sensor values).

3.4.5.2.1 Storage needs

According to the estimate in section 3.2.4, a fish farm consisting of 50 sea cages with 5 sensors each will produce 1 GB data during a month. To get a more representative number, this scenario was simulated and measured. The sensor simulator was configured with the right amount of sea cages and sensors, and set up to send data equivalent to one month of runtime. Results were as follows:

Size of database before simulation:
640KB

Size of database after simulation:
760,398MB

Amount of data produced:
759MB

The result shows that the estimate in 3.2.4 Data requirements was not that bad, and gave us a good number for the system requirements.

3.4.5.2.2 Response times

According to the quality requirements of the system, the response time shall not be over one second. To measure this, a timestamp was set before and after launching the command, and the difference were calculated. Because this value is difficult to display in the FiFaMoS Mobile Context Consumer, the *System.out.println()* method was used in the FiFaMoS Context Consumer. Response times for the FiFaMoS Mobile Context Consumer would probably be a bit higher than the ones stated in Table 26 because of the narrowband connection used. Various scenarios were tested, and the results are presented in the table. The tests assume that the sea cages send their sensor data to the context manager once a minute.

Table 26: End-to-end response time measurements in milliseconds. The number in () indicates the processing time in the context manager

Test number	Number of sea cages (each with 5 sensors)	Time for getting current sensor values for a sea cage	Time for getting the sensor value history for the last 24 hours	Time for getting the sensor value history for the last week
T1.1	5	240 (150)	190 (40)	512 (160)
T1.2	5	213 (115)	203 (29)	477 (212)
T1.3	5	253 (142)	186 (64)	540 (204)
T1.4	5	249 (143)	211 (51)	451 (254)
T1 average time		**239 (138) ms**	**198 (46) ms**	**495 (208) ms**
T2.1	25	480 (420)	180 (80)	460 (145)
T2.2	25	512 (449)	196 (91)	521 (163)
T2.3	25	529 (461)	201 (76)	488 (170)
T2.4	25	502 (444)	178 (97)	495 (144)
T2 average time		**505 (105) ms**	**189 (86) ms**	**491 (156) ms**

Test number	Number of sea cages (each with 5 sensors)	Time for getting current sensor values for a sea cage	Time for getting the sensor value history for the last 24 hours	Time for getting the sensor value history for the last week
T3.1	50	640 (552)	214 (109)	482 (182)
T3.2	50	701 (586)	205 (142)	563 (172)
T3.3	50	637 (549)	209 (120)	502 (170)
T3.4	50	608 (522)	195 (113)	498 (148)
T3 average time		**647 (552) ms**	**206 (121) ms**	**512 (168) ms**

To find out if the performance was dependent on the number of sea cages, three tests were carried out: One with 5 sea cages, one with 25 sea cages and one with 50 sea cages. From the test results it can be seen that the number of sea cages has got most impact on the context manager site when asking for the current sensor values. This is because this request triggers many database requests, and more instances in the database makes the responses slower. The transmission time (total time – context manager processing time) seems to be almost constant. This is because the amount of data transferred does not get affected by the total number of sea cages in the system.

Also when asking for historical sensor values, the total number of sea cages does not affect the performance considerably. Only data from one sensor is transferred, and the parameter that affects the performance the most is the chosen period. It can be seen that the last week request takes about three times as long as the last day request, and this is because seven times as many sensor values have to be transferred. As the numbers in the table show, the transmission time (total time – context manager processing time) is the most dominant part of the total time. The reason why the week request does not take seven times as long as the last day request, is that XML-RPC uses the GZIP algorithm to compress the response [52]. This way bandwidth usage is saved, and response times gets lower.

All of the response time measurements are under 1 second, and the quality requirements are fulfilled. However, with even more sensor value in the database, the *getHistoryForSensor()* method would probably generate a response requiring more than one second to transfer. This especially comes into effect on the mobile context consumer, which in most cases would use a GPRS narrowband connection.

3.4.6 Deployment

For the system to work, some of the elements in Table 27 must be present. The *required* column indicates if it is optional or not. Note that neither of the context consumers are required, but for the system to be useful, one of them must be used.

Table 27: System components

Component	Component type	Component description	Required
C1.1	Hardware	Computer for the context manager	V
C1.2	Software	APMS middleware context manager	V
C1.3	Software	FiFaMoS service	V
C2.1	Hardware	M2M GSM module with built-in or external GPS	V
C2.2	Software	FiFaMoS Context Source IMlet	V
C2.3	Hardware	GPS and various sensors or dummy sensors that are compatible with the inputs of the M2M module	V
C3.1	Hardware	Mobile phone/PDA with Internet connection supporting Java	X
C3.2	Software	FiFaMoS Mobile Context Consumer MIDlet	X
C4.1	Hardware	Computer running the FiFaMoS Context Consumer	X
C4.2	Software	FiFaMoS Context Consumer J2SE application	X

The deployment of the system is also modelled as an UML deployment diagram in Figure 38.

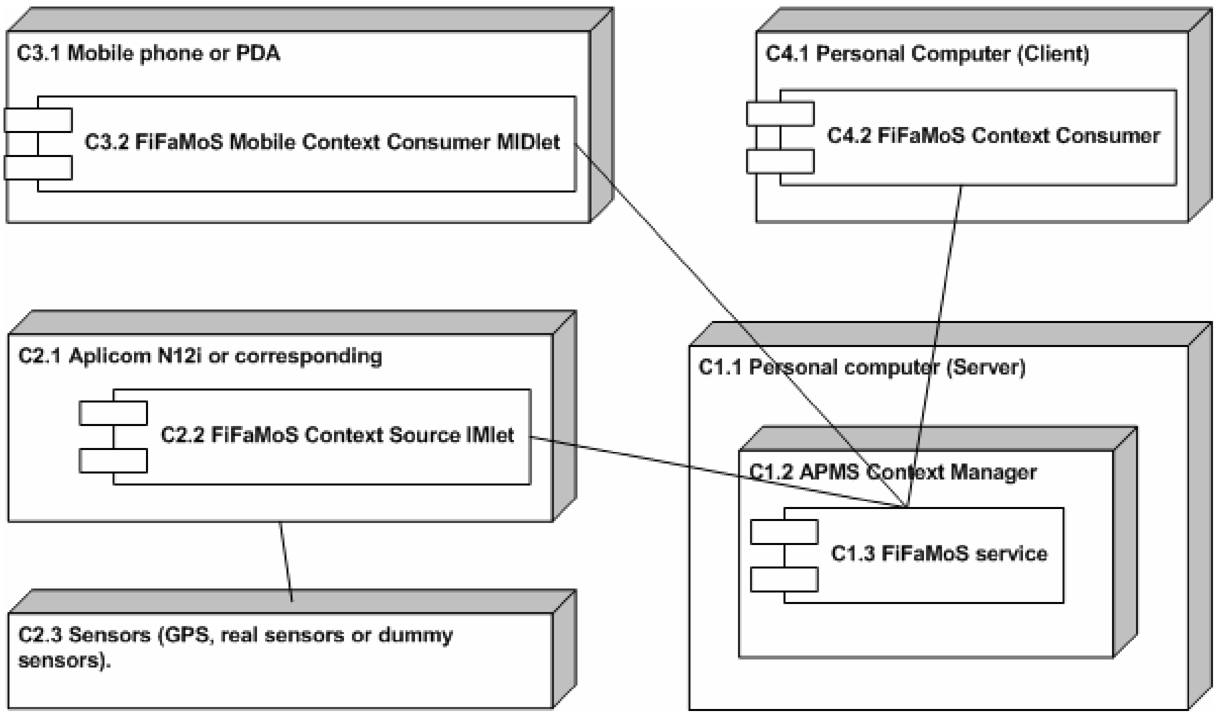

C3.1 Mobile phone or PDA
C3.2 FiFaMoS Mobile Context Consumer MIDlet
C4.1 Personal Computer (Client)
C4.2 FiFaMoS Context Consumer
C2.1 Aplicom N12i or corresponding
C2.2 FiFaMoS Context Source IMlet
C1.1 Personal computer (Server)
C1.2 APMS Context Manager
C1.3 FiFaMoS service
C2.3 Sensors (GPS, real sensors or dummy sensors).

Figure 38: Deployment view of the system

Since the whole system is implemented in Java, the computers running the different applications can use any operating system that supports Java technology. However, a runtime environment must be present.

3.4.6.1 Context manager

The APMS context (C1.2) manager needs a computer with a static IP address and port accessible from the Internet. The placement of this computer is irrelevant. All context inputs, outputs and processing are done on this computer by the APMS container and custom made services (C1.3). The database in the context manager stores wanted sensor values for historical purposes. The APMS context manager is not very greedy on resources. A relatively basic computer equipped with a gigahertz processor and 256MB RAM will have no problem running the context manager. The computer does not need an external database and web-server, since the APMS has got these integrated in the application.

3.4.6.2 Context source

To collect the different sensor data, and pass it on to the context manager, one or more M2M modules are needed (C2.1). There will be one module per sea cage. These modules have got several digital as well as analogue inputs and outputs that can be read and configured via Java.

The module communicates with the context manager (C1.2) via its GPRS or EDGE interface. For a fish farm surveillance system, various sensors (C2.3) are also needed. Since this is only a prototype, only dummy sensors will be used to input sensor data. If the system is to be used in a real fish farm, it is important that the sensors and modules have high industrial quality, and are able to cope with extreme humid conditions, and repetitive movements.

To get positioning information of the sea cages, a GPS receiver is needed. Some M2M modules have got a built-in GPS receiver, others can be connected to a standard GPS receiver via the serial interface on the module.

3.4.6.3 Context consumer

On the user side of the system (context-aware application in Figure 5), several options are possible. The system can be accessed via a J2ME mobile client (FiFaMoS Mobile Context Consumer, C3.2), or a computer running the PC client (FiFaMoS Context Consumer, C4.2). Most of today's computers will be able run the application; however the mobile client requires a mobile phone supporting MIDP 2.0. Both the PC client and the mobile client require some kind of Internet connection.

3.4.7 Encountered problems

Not everything has gone as smoothly as hoped during the project, and here some of the problems encountered will be described and discussed.

3.4.7.1 The APMS context manager

The APMS context manager was checked out from a CVS repository located in Tromsø. When building the context manager, errors occurred, and nothing worked. It was revealed that the jar-files were destroyed when checking out, and they had to be replaced by new ones.

When deleting packages in the APMS web interface, the context manager crashed. This was found to be a bug in the APMS system, and was fixed by the team that implemented it at the University of Tromsø.

When deploying the JAR-file as a service on the context manager, the context manager unpacks the JAR-file and loads the components. But it also generates new meta files, based on the ones in the JAR-file. This generation, however, has some flaws. At some occurrences, seemingly random, the generation of new meta files are erroneous. After some error searching it was revealed that there was an error in the *MetaDataParser.java* file in the APMS context manager. This error is described in detail in appendix B.9.

The Database abstraction in the APMS context manager do not support relations, so in the first place we used only one database table that described all attributes of the sensor. This solution was easy to implement, but it is not flexible, and it doesn't describe the semantics of the system well. Instead we decided to use the database model shown in Figure 21, and handle all relations manually. We used regular fields as foreign keys, and manually added the parent table's primary key to them. This causes some extra database load, but mostly only when adding table entries.

APMS comes with classes to insert, select and delete entries from the database, however, a class for updating tables was also needed. This was implemented and added as a part of our service package (See appendix B.6).

During the project, the APMS context manager has been updated many times. After installing a new version of the APMS context manager, the system did not work. When the database tables should be created, a number of exceptions were thrown. The error was that the primary key was set twice. This bug is described in more deeply in appendix B.8.

3.4.7.2 The Aplicom N12i Internet connection

During the first days of the project, there were some problems getting the Aplicom N12i on the Internet. It was revealed that the GPRS settings was not correctly set. The settings can be altered using the Aplicom N12i configurator (see appendix E.1), and the settings that worked in the FiFaMoS project are listed in Launching the FiFaMoS context source.

3.4.7.3 Communicating with the camera

To view video or pictures from the fish farm, a camera was needed. Most web-cameras have an USB interface, but since the FiFaMoS project uses only M2M modules, a camera with a serial interface was needed. TraceMe has an M2M based solution for transport firms with included camera. This camera was connected to the FiFaMoS M2M module, and we tried to get photos from it. First of all, the physical connection was not easy to accomplish. The camera had a 4-pin plug that is commonly found on old soundcards. To make this fit to the 9-pin D-Sub on the M2M module, an adapter had to be made (see appendix D.2). Also we had to search the Internet for the pin configuration of the camera.

When the camera was connected, we tried to read serial data from it, but it turned out to be dead. After reading documentation found on the internet [43], we understood that several hexadecimal commands had to be sent to the camera to get it working. Some of the commands were SYNC, ACK, set package size, get picture, etc. First of all, the SYNC command should be sent several times, until an ACK command was returned by the camera (as shown in Figure 39).

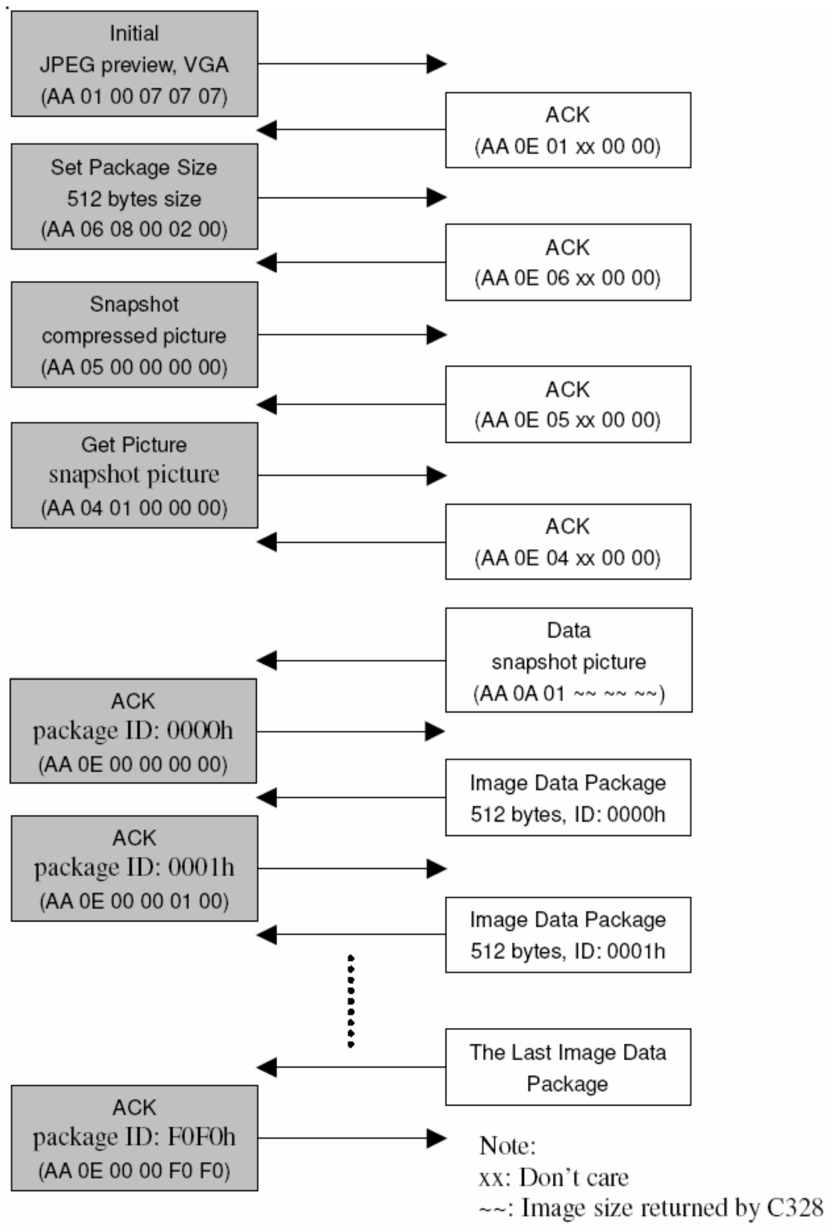


Figure 39: Camera commands

We tried several parameter and pin configurations, but we could not get the ACK command from the camera. We also tried to send the SYNC command to the serial port using different formats, but none worked. After many attempts, we decided to use our time on more important functions in the system, and put the camera away.

3.4.7.4 Communication with the GPS

An M2M module with integrated GPS functionality was not available, so an external GPS receiver had to be connected via the serial interface on the M2M module. The GPS receiver used was a Magellan SporTrakColor that was able to output the GPS data on the serial port periodically using the NMEA 0183 format.

Figure 40: The GPS receiver we used: Magellan SporTrakColor

Connecting the GPS to the module was not as easy as expected. Both the GPS and the M2M module had a female serial port connector, so an adapter had to be made. The adapter consisted of two male 9-pin D-SUB serial port connector connected with crossed wires (see appendix D.3). When the adapter was in place, and the communication worked, we tried to use the NMEA 0183 parser integrated in the M2M module's API. This did not work at first, so we considered making our own parser. But after trying different NMEA configurations on the GPS receiver, we made it work. The NMEA setting on the GPS receiver was set to 'NMEA V2.1 GSA'.

3.4.7.5 Timestamp problems

The sensor value timestamps (generated at the context manager) gets too big for the FiFaMoS Mobile Context Consumer. This is stored as the primitive data type long indicating number of milliseconds since January 1st 1970, and was fine on the context manager side of the system. However, when the time stamp arrived at the FiFaMoS Mobile Context Consumer, it was incorrect, and the new date we created did not match the original one. We have been searching the web for solutions, but with no luck. The way we made it work was to divide every timestamp by 1000, and multiply it by 1000 before restoring the date from the timestamp.

4 Discussion

During the project, several decisions had to be made. That included both software and hardware options in addition to the architectural and implementation choices. Section 4.1 will look at how well the APMS context manager is suited for the FiFaMoS project, section 4.2 will discuss the data transmission options, section 4.3 will discuss the storage of data, and at last the hardware options are discussed. In addition the risk analysis from the prestudy section will be evaluated.

4.1 APMS as a context manager

To make the sensor information (context) available to the clients, a context manager is needed. The main task of the context manager is to get sensor information from the fish farm, store it in persistent memory, and send it out to the clients that need it. Originally, it was intended to use the context management system from the Akogrimo [2] project. But after some research, it was revealed that other context management solutions that suited the FiFaMoS project better existed. One of these is the APMS context manager.

The main reason why Akogrimo is not suited for the FiFaMoS project, is its complexity. With its various interfaces for context and presence information its primary target tasks are advanced scenarios like e-health and e-learning. The simple objectives of gathering and distributing sensor information of fish farms do not need such advanced functionality.

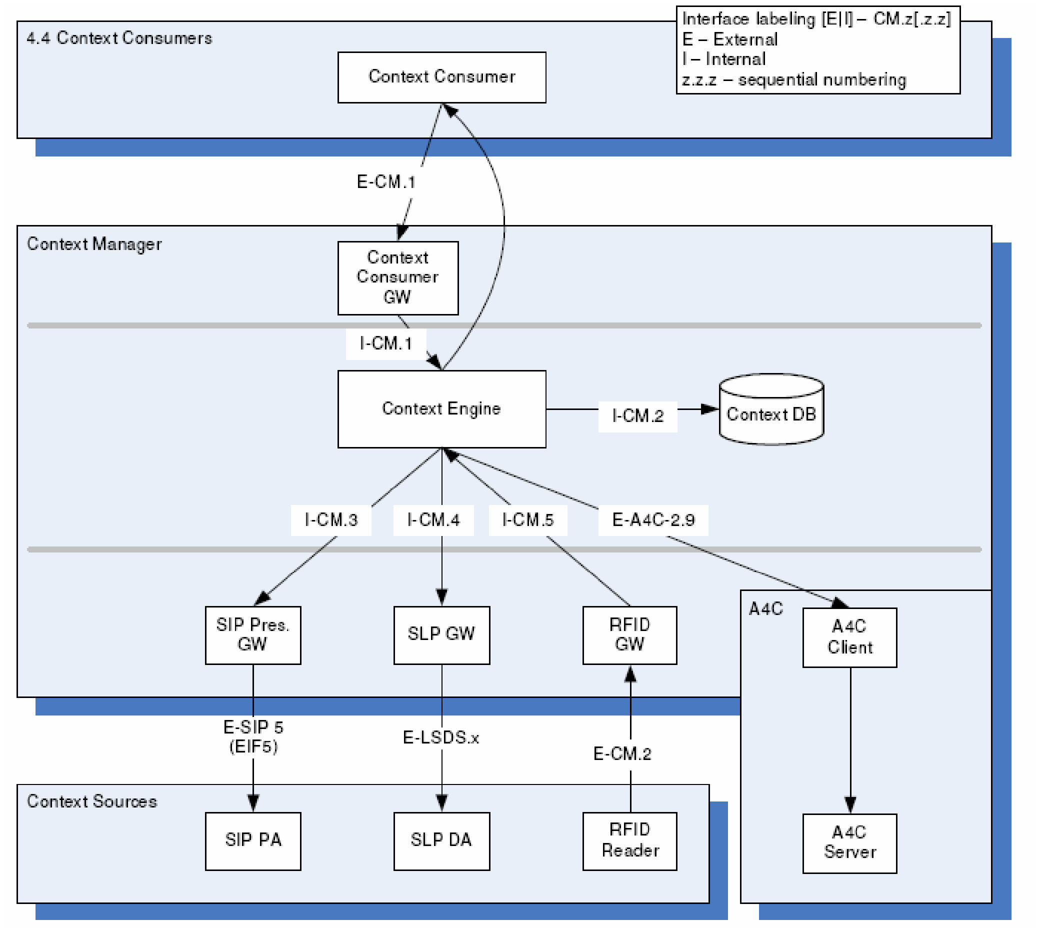


Figure 41: The Akogrimo context manager architecture

As seen in Figure 41, the context source interact with the context manager via either SIP SIMPLE, Service Location Protocol (SLP) or via a proprietary RFID protocol. These interfaces are not well suited for transferring sensor values from M2M modules, so an extra gateway would have to be created in the context manager.

The system is also based on subscribe/notify mechanisms, that require the context consumer to have a web service interface. This is difficult to accomplish on a mobile context consumer, and would require an extra node between the context manager and the context consumer. According to one of the design approaches of the FiFaMoS system, the architecture shall be as simple as possible. Adding extra nodes would violate with this approach.

However, the Akogrimo would offer advantages as well. The complexity makes it possible to add further functionality, like RFID-support, SIP sessions, and authentication mechanisms. In addition, the subscribe/notify messages enable the context manager to take initiative to communication towards the clients. Because the clients do not need to continually get the sensor values from the context manager, they can be notified only when a sensor value has changed. This saves bandwidth usage.

Another advantage with the Akogrimo system, is that SOAP web services are used. This makes it possible to transfer objects in their original shapes, not needing to convert them into XML or other string representations.

As mentioned, the context manager used in the FiFaMoS system, is the APMS context manager. This is a component based middleware system with built-in support for various binding types and persistent storing. By implementing some input- and output components, a service can be established, and accessed by context sources and consumers. The ease of implementation, and simplicity of use made this the chosen context manager in the FiFaMoS system. The binding- and storage management makes it well suited for sensor based surveillance tasks like this project, and it does not have much functionality that is not needed.

4.2 Data transmission

Several aspects have to be discussed when it comes to data transmission. How should data be represented? How should it be transferred, and which physical connection should be used? These topics are discussed in this chapter.

4.2.1 Binding types

To allow communication between the nodes in the system, a binding must be established. This can be done in several ways using various technologies like TCP socket, UDP socket, RMI, CORBA and web services.

The APMS context manager has built-in support for multiple binding types, including web services (XML-RPC), TCP socket, UDP socket and Java RMI. Using sockets or RMI makes a Connection Oriented Architecture, where few connections are made. This leads to a low amount of overhead and great performance. The web service option creates a new connection every time data is needed, which leads to a higher amount of overhead [24]. Still, it is a benefit that the connection is not established when no data is needed.

The main reason for using web services is to get a Service Oriented Architecture (SOA) where everything is very loosely coupled. This makes it easy to include various clients in the system, and firewalls in the Internet will be traversed if a well-known port as 80 is used. The drawback is that the context manager needs a web server to receive commands, but that is not a case in this project since the APMS already include the Jetty web server. Another drawback is performance. Web services are many times slower than a socket, RMI or CORBA connection since the web server adds another "link", and overhead to the messages. However, if the messages are small in size, the performance problem will not be that significant.

The web service that is supported in the APMS system is XML-RPC. This is SOAP's predecessor, and is more lightweight and easy to use. One could argue that the technology is outdated, and is on it's way to the grave, but it still does a great job if the data types are of the primitive kind (see section 2.2.5). The lightness of the server and client also makes it easy to implement in mobile devices based on J2ME.

In the FiFaMoS system, most responses are vectors containing objects. If the APMS supported SOAP, these objects could be transferred like they were, but since XML RPC only support more primitive data types, the responses have to be transferred as a string. This problem is described in the next section.

4.2.2 Data representation

When transferring data (fish farm information and sensor values), data has to be coded into a string for transmission via XML-RPC. Two ways of doing this are XML coding and comma/colon separated values (CSV). XML coding makes the information easy to understand for humans, and the system gets flexible and easy to alter. The drawback is that the XML tags add a lot of overhead. Using comma separated values minimizes overhead, but the system becomes very static, and difficult to modify. The format of the comma separated values has to be agreed on by both sides of the system.

In FiFaMoS, it is used CSV to code RPC-responses containing sensor values, while responses containing fish farm information (*getFishFarmAsXML*) are coded as XML. Using XML for this makes it possible to receive the whole fish farm setup with the use of only one RPC-request. To illustrate the different data sizes and needs for requests, an example is introduced. This example has three frames, each with three sea cages:

The GUI tree for navigation between frames and sea cages is to be filled. This can be done either with the *getFishFarmAsXML(),* or by first getting the frames as CSV and then the sea cages as CSV. Using the XML method, a response like this is generated:

```
<xml>
 <frame>
   <id>9</id>
   <location>Frame #0</location>
   <description>Telenor test cage</description>
   <mobile>91808620</mobile>
   <email>jagrodal@broadpark.no</email>
   <seacage>
     <id>35</id>
     <latitude>63.49638888888889</latitude>
     <longitude>10.399166666666666</longitude>
     <description>Sea cage #0-0 (Aplicom 12)</description>
     <imei>352540000046895</imei>
     <frameid>9</frameid>
   </seacage>
   <seacage>
     <id>36</id>
     <latitude>No GPS</latitude>
     <longitude>No GPS</longitude>
     <description>Sea cage #0-1</description>
     <imei>0</imei>
     <frameid>9</frameid>
   </seacage>
 </frame>

 <frame>
   <id>10</id>
   <location>Frame #1</location>
   <description>Telenor test cage</description>
   <mobile>91808620</mobile>
   <email>jagrodal@broadpark.no</email>
   <seacage>
     <id>38</id>
     <latitude>No GPS</latitude>
     <longitude>No GPS</longitude>
     <description>Sea cage #1-0</description>
     <imei>2</imei>
     <frameid>10</frameid>
   </seacage>
   <seacage>
```

```
      <id>39</id>
      <latitude>No GPS</latitude>
      <longitude>No GPS</longitude>
      <description>Sea cage #1-1</description>
      <imei>3</imei>
      <frameid>10</frameid>
    </seacage>
  </frame>
</xml>
```
Code 34: Excerpt from the getFishFarmAsXML() response, showing two of three frames.

The response represented as a string has a total amount of 2316 bytes. Since the XML-RPC response is compressed using HTTP compression (GZIP), the amount of transferred bytes is lower. This was measured to 521 bytes.

Using CSV will first require one RPC-call to get all frames, and then one call per frame to get the sea cages. The *getFrames()* response is as follows:

```
9:Frame #0:Telenor test cage:91808620:jagrodal@broadpark.no;10:Frame
#1:Telenor test cage:91808620:jagrodal@broadpark.no;11:Frame #2:Telenor
test cage:91808620:jagrodal@broadpark.no
```
Code 35: Response from the getFrames() method

An the three getSeaCagesForFrame(frameID) requests produce these responses:

```
35:63.49638888888889:10.399166666666666:Sea cage #0-0 (Aplicom
12):352540000046895:30:9;36:No GPS:No GPS:Sea cage #0-1:0:30:9;37:No GPS:No
GPS:Sea cage #0-2:1:30:9

38:No GPS:No GPS:Sea cage #1-0:2:30:10;39:No GPS:No GPS:Sea cage #1-
1:3:30:10;40:No GPS:No GPS:Sea cage #1-2:4:30:10

41:No GPS:No GPS:Sea cage #2-0:5:30:11;42:No GPS:No GPS:Sea cage #2-
1:6:30:11;43:No GPS:No GPS:Sea cage #2-2:7:30:11
```
Code 36: Response from the getSeaCagesForFrame(frameID) method

The total amount of bytes for these responses are 590, and they were compressed to a total amount of 355 bytes. This means that CSV representation in this example reduces the amount of transferred data to 68%. However, the need for invoking the methods multiple times, leads to a response time higher than with XML representation. Adding more frames or sea cages would increase the number of requests, and the respones time further.

In addition, all commands sent from the context source, and the add/remove/update commands from the context consumer are coded as XML. This is done by generating an

XML-document containing a method-tag, and fields for various attributes. This makes it possible to run multiple internal methods via only one invoked RPC-method, and the interface won't have to be changed when adding new functionality.

The reason why XML representations are not used when getting the latest sensor values, alarms, is that the time span of this project did not allow the implementation. CSV is easy to implement, and by adding an object constructor that takes the comma/colon separated string as input (see Code 37), objects can easily be created from the string.

```
public SeaCage(String input) {
  StringVector sv = new StringVector(input, ':');

  id = Integer.parseInt(sv.stringElementAt(0));
  latitude = sv.stringElementAt(1);
  longitude = sv.stringElementAt(2);
  description = sv.stringElementAt(3);
  imei = sv.stringElementAt(4);
  refresh = Integer.parseInt(sv.stringElementAt(5));
  frameid = Integer.parseInt(sv.stringElementAt(6));
}
```

Code 37: The SeaCage object constructor that takes a CSV string as input.

The only method that should use CSV in its response is the *getHistoryForSensor(sensorID).* The number of sensor values transferred could be very high, and adding XML tags for the value and timestamp would dramatically increase the amount of data transferred. This would also make the response times higher. If XML should be used, it would be essential to use a format as short as possible. This could be done by using empty tags and short names (e.g. <sv sid=5 v=345 t=544322 /> where sv means sensor value, sid equals sensorID, v value and t is the timestamp.

The sensor values itself can be represented in many ways as well. When reading an input on the M2M module, the mV value is returned. This value is interpreted, based on the sensor type, in the input component, and then stored in the database. It would also be possible to interpret the value on the M2M module, or in the clients, but it was found more convenient to do it in the input component. Thus placing most of the logic centralized in the system.

To represent the position revealed by the GPS, several formats can be used. The most known way is the usage of latitude and longitude parameters. This works well in most cases, but in this thesis, it is wanted to compare different positions. Using latitude and longitude introduces the problem that the physical length of a latitude degree is not constant, but is relative to the length away from the Equator. These parameters is therefore not suited when distance between locations shall be reckoned. Another way of representing positions that is more suited for these tasks, is the UTM system (see 2.2.7.2). This introduces positioning information in metres, and makes calculation of distance an easy task. In the FiFaMoS system, GPS information from the M2M module is stored in the database as latitude and longitude, and before verifying the current position of the sea cage, the positioning data is converted into the UTM system. The distance between the initial position and the current position is then calculated, and an alarm is triggered if the sea cage is out of position.

4.2.3 The off-shore Internet connection

To transfer the sensor data from off-shore to on-shore, a connection is needed. Since the FiFaMoS system is Internet based, an Internet connection is to be preferred for collection sensor data. The solitary location of the sea cages make this a challenge, and since the sea cages aren't connected to on-shore by wireline, only wireless alternatives are usable.

The M2M modules used in the project only support GPRS and EDGE as data transmission link. However, if the fish farm is not too far away from the mainland, other hardware and technologies could be used. WiMax is an option that offers high data rates, and it is also cheaper to use than GPRS. According to [41], AT&T trials were able to get WiMax based communication ranges up to 5 miles. This will probably be good enough to reach many fish farms. With bandwidths up to several Mbps, this connection is an optimal solution if real-time video surveillance is needed. The drawback is that M2M modules supporting WiMax technology is currently not available, so an industrial computer would have to be situated at the frames of the fish farms.

4.3 Data storage

When sensor data shall be stored to enable presentation of historical values, several aspects will have to be discussed. One important case is how much data is supposed to be stored.

If for instance 100 sea cages, each with 10 sensors, update their sensor data every ten seconds, there will quickly be generated a huge amount of data. After some time the context manager database will fill up, and either make the system unstable or crash the whole system. To avoid this, without increasing the refresh interval, an algorithm for deleting redundant sensor values must be implemented. This could be a program module that continually checks the values related to each other, and deletes values that are very close to its predecessor. It is also possible to utilize existing algorithms like the Kalman filter. This is an efficient recursive filter which estimates the state of a dynamic system from a series of incomplete and noisy measurements. This can be utilized to replace a whole lot of sensor values with the only values needed for representing the curve.

Storing all sensor values in the database also affects bandwidth usage and client response times. When a client asks for historical sensor values in a given period, all sensor values in the period are returned, and this could be a relatively high amount of data. This could be a problem especially for the FiFaMoS Mobile Context Consumer that most often will use the GPRS network. If a sea cage updates its sensor data once a minute, the following amount of data will be generated per sensor during a week:

*Amount of data per sensor = amount of data per sensorvalue * number of minutes in a week = 100 bytes * (60 * 24 * 7) minutes = 1080000 bytes/month = 1054,69 KB/week = 1MB/week*

If a client asks for the sensor history from the last week, all this data will have to be transferred. Due to the integrated GZIP compression algorithm used in XML-RPC, the total amount of data will shrink with a factor of about 0,4 [52]:

*Amount of data transferred = total amount of data * compression factor = 1054KB * 0,4 = 412 KB*

and that could take a while if the client uses a GPRS Internet connection (the following example assumes that the GPRS connection is capable of transferring 5KB/sec):

Transfer time = Amount of data to be transferred / line capacity = 412 KB / 5KB/sec = 82 seconds = 1 minutes and 22 seconds

A response time over one minute when asking for the last week's sensor values is not acceptable. To get this better, two things can be done: Reducing the amount of data stored, as mentioned over, or only reducing the amount of data transferred by using an algorithm similar to the one that should be used for minimizing the storage of data. The implementation of an algorithm like this is found to be outside the scope of this thesis, and is therefore placed in the further work section.

4.4 Sensor value checking

Sensor values from the fish farm have to be checked whether they are within the allowed scope of the sensor. This could be done in several ways, including on-the-fly checking, explicit checking by a processing component, or checking by the use of the Drools engine in the APMS middleware.

First a processing component was made that fetched sensor values from the database in given intervals and checked if they were OK. This causes too much database load, and the solution was turned down. Then the Drools engine included in the APMS engine were tried out. It was revealed that this feature was not yet finished in the APMS system, so it was not possible to use it. The solution that was used in the end was on-the-fly checking by the input component. This way the sensor value is checked when arriving, and the database load is minimized. In the future it could be useful to utilize the Drools engine. This makes it possible to alter the rules via the APMS web interface (see appendix C.2).

4.5 Context source hardware options

There exist many solutions for gathering off-shore sensor information and passing them to the context manager via the Internet. Based on power needs and size, M2M solutions are a good choice. However M2M will bring some limitations, both in number of sensors and sensor connectivity interfaces. Here we will discuss some sensor data acquisition solutions.

4.5.1 Aplicom L4002 M2M module

The Aplicom L4002 is designed for wireless telemetry and vehicle tracking systems, and has a built-in GPS. With the help of a waterproof case, this module will suit the FiFaMoS system well.

Figure 42: Aplicom L 4002 M2M module

The Aplicom L 4002 module is programmable in java (J2ME MIDP 1.0, IMP), and it has GSM and GPRS/EDGE mobile interfaces. It also has serial ports that can be used to upload applications to the module, or to connect peripheral equipment. This unit consists of the Aplicom N12 M2M module, bundled with a GPS receiver, so it is basically the same setup as the one used when developing FiFaMoS. The main difference is that it has got a suited steel case. Figure 43 shows the module's interna architecture.

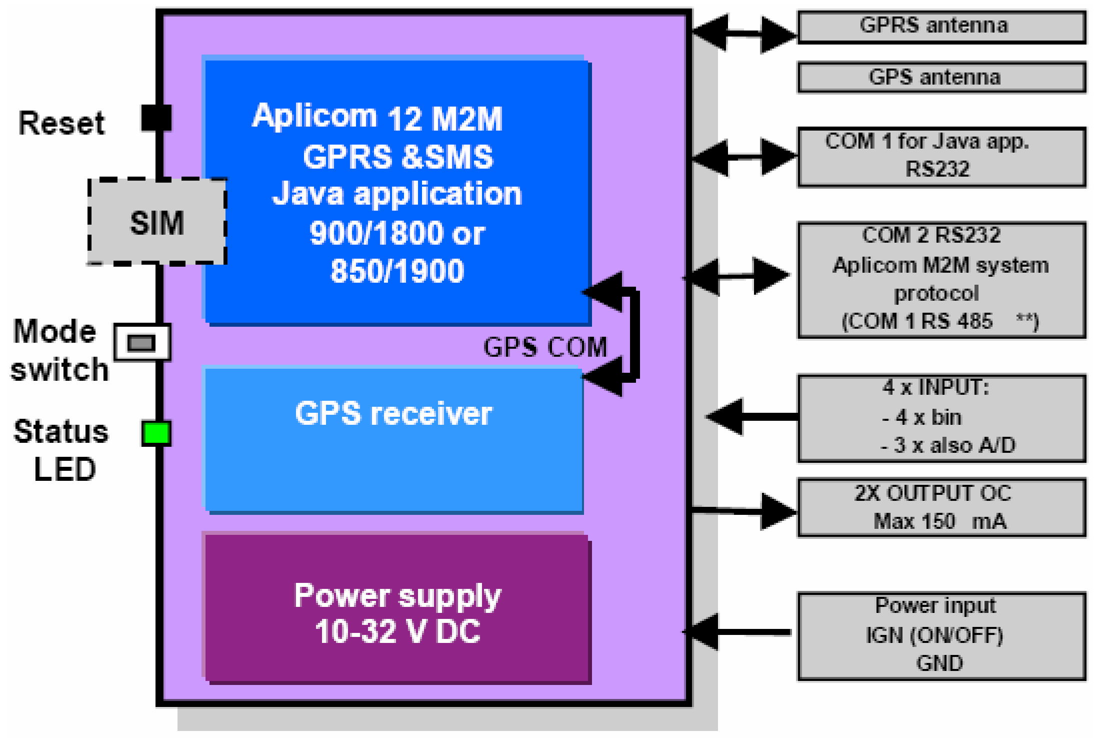

Reset
SIM
Mode switch
Status LED
Aplicom 12 M2M GPRS &SMS Java application 900/1800 or 850/1900
GPS COM
GPS receiver
Power supply 10-32 V DC
GPRS antenna
GPS antenna
COM 1 for Java app. RS232
COM 2 RS232 Aplicom M2M system protocol (COM 1 RS 485 **)
4 x INPUT: - 4 x bin - 3 x also A/D
2X OUTPUT OC Max 150 mA
Power input IGN (ON/OFF) GND

Figure 43: Aplicom L 4002 architecture

The inputs and outputs are described further in Table 28.

Table 28: Aplicom L 4002 outputs and inputs

Input/output	Description
COM 1	Serial port used for uploading applications to the module
COM 2	Serial port reachable from the java runtime. This could be used to transport sensor information or control information
4 x INPUT	Four inputs. Three of these can be configured as analogue inputs.
2 x OUTPUT	Two open collector analogue outputs.
Power input	10 – 32 V DC
Reset	Reset switch
SIM	SIM card slot
Mode switch	The module can be operated in two different modes: system mode 1 and system mode 2. We will be using system mode 1, because the AT commands enabled in system mode 2 is not needed in our system.
Status LED	LED that indicates the status of the module.

4.5.2 Teltonika T-Box/GPS M2M module

Teltonika T-Box/GPS is also an M2M module that has got a built-in GPS. The module has got 4 digital inputs, 4 digital outputs and 3 analogue inputs. Since the module is based on the same chip as the Aplicom L4002, the same Java IMlets can be used. However this module has some extra features, including RFID reader and built-in temperature sensor.

Figure 44: Teltonika T-box/GPS

T-Box/GPS has integrated rechargeable Li-Ion battery and special controller for power management. This feature enables operation without using external power supply (up to 5 hours).

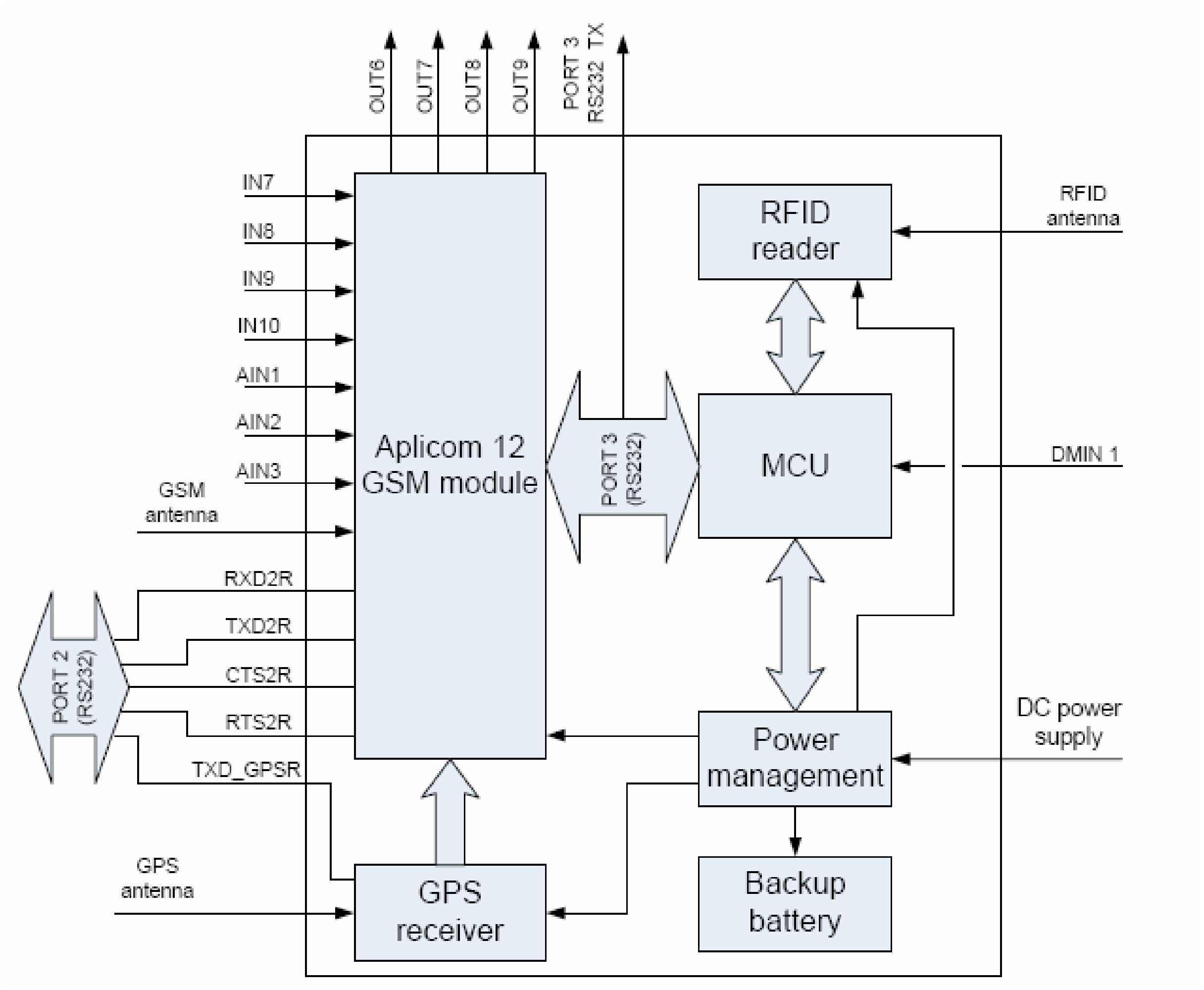


Figure 45: T-box/GPS internal architecture

The inputs and outputs are described in .

Table 29: Teltonika T-Box/GPS outputs and inputs

Input/output	Description
PORT 2	Serial port used for uploading applications to the module.
PORT 3	Serial port reachable from the java runtime. This could be used to transport sensor information or control information
IN7-10	Four inputs. Three of these can be configured as analogue inputs.
2 x OUTPUT	Two open collector analogue outputs.
DC Power supply	Input for supplying power to the GPS receiver and M2M module.
RFID	Antenna for the integrated RFID reader.

Input/output	Description
antenna	
GSM antenna	Antenna for the GSM and GPRS connunication
Mode switch	The module can be operated in two different modes: system mode 1 and system mode 2. We will be using system mode 1, because the AT commands enabled in system mode 2 is not needed in our system.
GPS antenna	Antenna connector for the internal GPS receiver

4.5.3 Other M2M solutions

Other M2M hardware like the BlueTree 5400 GPRS and the TraceMe solution from KCS have similar functionality and connectivity but these do not allow you to use your own applications. The functionality and actions are user set via special software, and the units communicate via special servers to the client software. This makes the modules incapable of communicating via our context management system, and the modules will not be possible to use in our project.

Figure 46: BlueTree 5400 GPRS M2M module

4.5.4 Industrial computer with GPRS modem

An alternative to M2M modules are industrial PCs. These have a compact cast aluminium chassis that provides great protection from shock, vibration, dust, cold and heat while acting as a functional heat sink to ensure lower temperature operation. An industrial PC can do the same as normal personal computers, but they often have more inputs and outputs. This opens for connecting more external peripherals as sensors and sensor controllers. Keyboard and a monitor will have to be connected as well.

Figure 47: Advantech industrial computer [28]

The draw backs are that they cost a lot more than an M2M module, and size and power needs make this a demanding solution. Also an industrial PC is a lot more complex than an M2M module, and more things can go wrong.

Still, having the opportunity to connect almost anything (including cameras, weather stations and other USB-based equipment) is a great advantage. A solution where each sea cage have an M2M module, and in addition the fish frame have an industrial PC would be near optimal. The industrial PC could then have the responsibility for gathering data common for all the sea cages in the frame. This could be video surveillance and weather station data. Of course the computer needs an internet connection. This could me achieved by a GPRS modem, or maybe WiMax connection if present off-shore.

4.5.5 Summary

M2M modules like the BlueTree 5400 GPRS and the TraceMe solution are pre-programmed, and only need proper configuration to work as surveillance or tracking systems. However, making them work with existing middleware solutions like the APMS system is impossible since new applications cannot be uploaded to the modules.

The Aplicom M2M L4002 or the Teltonika T-box/GPS are better choices since they are programmable in Java, and can be customized to interact with almost any kind of middleware system. The drawback is that the number of sensor interfaces are limited, and peripheral equipment using for instance USB cannot be connected.

If a lot of advanced peripheral equipment needs to be connected to the context source, an industrial computer is a good choice. Since these have a lot of interfaces and interface types, almost anything can be connected. The drawback is that an industrial computer is more complex and the probability of failing is higher.

Because only a small amount of sensors had to be connected/simulated in this proto type, an M2M module was the best choice. Connecting a GPS to the serial port, and for instance three sensors to the analogue inputs will make it possible to monitor the most important parameters in a sea cage.

4.6 Other aspects of the system

To get this system up and running off-shore, some other aspects will have to be figured out, and some hardware will have to be realized. This is especially adapters for sensors that can transform the sensor output to a voltage level that the M2M module can interpret. However, the M2M modules and sensors need power to work, and this could be a problem on smaller off-shore installations without power supply.

A solution could be to equip each M2M module with solar cells and a battery pack, or possibly have a solar cell and battery pack for the whole fish frame. Also cabling between the power source, M2M modules and sensors is a matter that will have to be done professionally. The extreme weather conditions off-shore set high demands for cabling and sensor equipment.

These are all tasks that do not fit this thesis, and will have to be figured out by specialists later. That if the system is supposed to be realized. Another obstacle is today's pricing models on mobile phone subscriptions. Each M2M-module (each sea cage) needs a SIM-card to be able to transfer the sensor data to the context manager. With high monthly fees, a solution with many M2M modules would become very expensive in use. Using twin SIM-cards helps a bit, but the expenses are still very high. Hopefully the future will bring cheaper mobile phone subscriptions customized for M2M applications. This will probably boost the use of these kind of services, hopefully including the FiFaMoS.

4.7 Risk analysis evaluation

To evaluate the risk analysis performed in the prestudy, the elements of risk are repeated. Based on the outcome of the project, each element of risk is given comments.

1. The M2M technology used is new and not that much tried and tested
2. The APMS context manager is under development, and bugs can prevent our system from working properly
3. Needed expertise and resources can be difficult to obtain
4. The system implementation can be so unstable that the uptime for the system will not be good enough
5. Low system security will prevent the system from being useful
6. The M2M module does not have enough functionality to manage all the tasks mentioned in the requirements

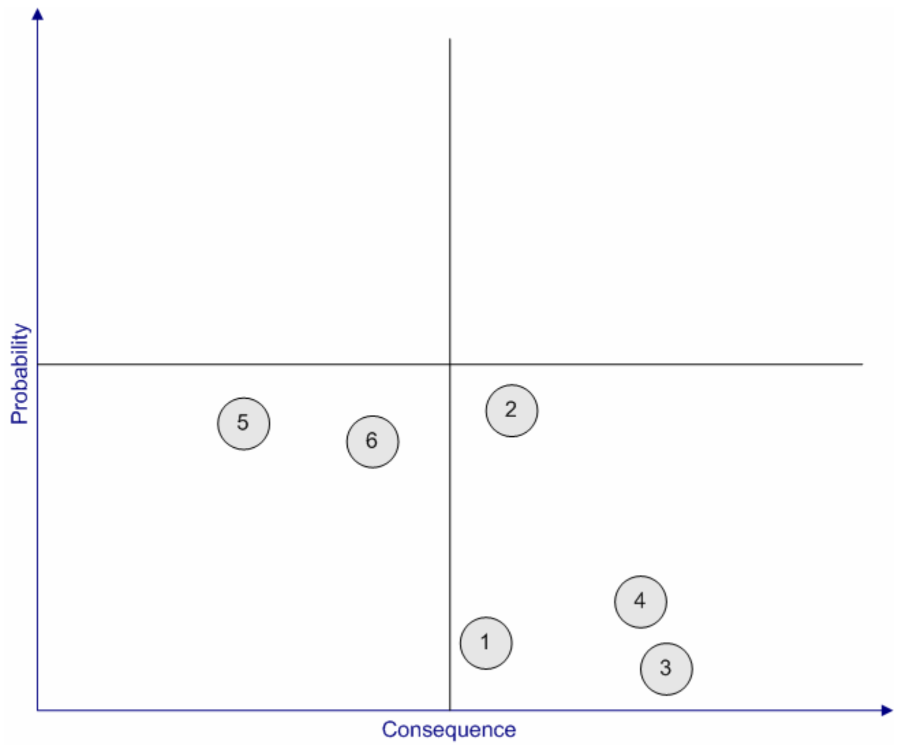


Figure 48: Risk elements diagram

4.7.1 Comments

The first element of risk is that the M2M technology is new and not that much tried and tested. This element had low probability and medium consequence. During the project it has not been much problems with the M2M technology itself. Much coding tips has been found on the Internet, and most problems got a solution in the end. The low probability of this risk element was right.

The second element of risk claimed that the APMS middleware is under development, and bugs could prevent our system from working properly. This was the element with the highest probability, and that was correct. Several bugs was found, and both we and the development had to make improvements. Luckily no serious bug occurred, and the APMS context manager could be used in the system.

The third element of risk said that it could be difficult to obtain the needed expertise and resources. This element had low probability and high consequences. When problems occurred during the development process, there was always someone to ask, either at Telenor R&D, or at the APMS development team via mail. The low probability of this risk element was right.

The fourth element of risk mentioned that a bad system implementation could make the system uptime too low. This element has low probability and medium consequences. Due to the limited time span of this project, long term testing has not been possible to perform, but we got the system running for several days without errors.

The fifth element of risk claimed that low system security would prevent the system from being useful. This element has medium probability and low consequences. AAA mechanisms have not been implemented, so the security of the system is not optimal. Since this only is a prototype, the security is not very important, and the low consequences prevents the system from being a failure.

The last element of risk was that the M2M module does not have enough functionality to manage all the tasks mentioned in the requirements. This element has medium probability and medium consequences, and has slightly been carried into effect. We was not able to make the M2M module read the data from the camera, so the requirement specification had to be re-evaluated.

5 Conclusions

A context-aware service (FiFaMoS) for the aquaculture industry makes it possible to continually monitor what is happening at fish farm, and to act on basis of the context information. The project has not been focusing on sensor integration, but with the help of some hardware, it is possible to connect most sensors to the system This way the whole fish production process can be monitored, and actions can be taken when abnormal conditions occur. Using a monitoring system also minimize the need for visiting the fish farm, which is a great advantage since fish farms tend to be situated far from land these days. Because the increased monitoring and control of various parameters enables the fish farmers to keep the fish healthy, the general quality of the fish is greatly increased. In addition the environmental damages will be lowered due to tracking of lost sea cages, and optimal feeding routines.

The tracking is realized by the use of GPS receivers on each sea cage. The GPS functionality provides a great insurance for the fish farmers as they will receive an alarm if a sea cage works loose and drifts away. By using the FiFaMoS Context Consumer, the position of the sea cage will be shown in a map, and the sea cage can be retrieved. All other sensor information have alarm functionality as well. If a sensor value is higher or lower than the preset limits, an alarm will be generated. The alarm contains all the necessary information needed to locate the problem. This includes which sea cage it is, which sensor it is, the sensors limits and the sensed value, and finally a timestamp telling when the alarm was generated.

The FiFaMoS system utilizes a context manager called APMS described in section 2.2.1. During this thesis, the APMS has been thoroughly investigated, and found to be perfectly suited for this type of context-aware application. By implementing various components, a service can be deployed on the context manager. Context sources (In this thesis the FiFaMoS Context Source) can feed the service with sensor data, and context consumers (in this thesis the FiFaMoS Context Consumer) can monitor the acquired data, both current and historical. The service components consist of an interface and an implementation of the interface. Methods in the interface are made available via different types of bindings such as RMI, web services and sockets. Web services are used in this system to get a flexible and loosely coupled architecture. The combination of the functionality of the different components make up the service. By using APMS, a centralized and flexible architecture is achieved. This, together with web services, makes deployment of new functionality and development of new clients very fast and easy.

The data representation has been tested with both comma separated values (CSV) and XML (section 4.2.2). It's obvious that CSV produces less overhead when transferring, while XML is much more flexible and user friendly. Using XML also makes the development of new clients easier, as the format does not need to be predetermined. However, when getting historical values for a sensor, XML is not very suited. This is because the large number of data entities that will be transferred will require a huge amount of overhead as each sensor value needs a separate XML-element or attribute.

The amount of storage space that is needed depends on three factors; how many sea cages with how many sensor are in the system, how often shall sensors be sampled, and for how far back in time should historical values be stored. In section 3.4.5.2.1 it's measured that with 50 sea cages with 5 sensors each and sensor values are stored once a minute, it will be produced 760MB of data during a month.

The response times measured in section 3.4.5.2.2 has shown that the system is very fast, and that it scales well. This is very important in connection with expansion of the fish farm, as it's preferred that adding a frame should not increase the response times of the system. However, requesting historical sensor values for a long period of time generates a lot of data, and this would make the response times rather high, especially on the FiFaMoS Mobile Context Consumer. Therefore it is desired to have a data reduction algorithm that can remove redundant sensor values in the database (see section 4.3).

5.1 Future work

This section describes what can be done to make the system even better. That includes both modifications and supplements to the implementation and hardware.

The system has not yet been tested in an off-shore environment. As described in 2.2.2.1, the sensors that already are situated at the sea cages are not compatible with the inputs of M2M modules, so to enable interworking, hardware will have to be developed and used. This is not a very demanding task, and could easily be accomplished by a firm with the needed knowledge and equipment. It is also possible that some of the larger sea cages already have got sensors connected to an industrial computer. By multiplexing all the sensor information by using a proprietary RS-232 interface, and seding it to the M2M module, the FiFaMoS system could enable real-time wireless monitoring and provide for accurate positioning information from the sea cage.

When viewing historical values over a long period, a lot of data will have to be transferred, and a huge amount of sensor values will be displayed in a relatively small area in the graph diagram. This results in high response times, and a rather broad graph line where the value is difficult to interpret. By using a Kalman-filter or an equivalent algorithm on the context manager side, the sensor values can be limited to a smaller set of values that represent the curve in a better way. This also reduces the number of values that have to be transferred, so bandwidth usage will be saved and response times will be lower. By using the algorithm directly on the database, it could be used for deleting redundant values, and storage needs could also be saved.

Video surveillance is a very useful function in a fish farm, both under and over water. Since the camera that was used in the FiFaMoS system was not able to communicate with the M2M module, another camera should be tried out. There exist many cameras that are built for monitoring of fish farms, but these are mostly analogue, and can not be easily included in the FiFaMoS system. However, if a capturing unit is connected between the camera and the M2M module, it should be possible to digitalize the image and send it over the Internet to the APMS context manager. How and if this can be done can be a subject of further investigation. Bandwidth requirements will also have to be investigated in case of real-time video surveillance.

The system would probably benefit from a user authentication and authorisation system, where a logon routine would have to be done before using the system. This would allow users with administrative rights to alter the fish farm configuration, when operators only could see the sensor information. It would also prevent unwanted users from accessing the information in the system. An authentication/authorisation system would require an update of the database, where users and user rights are represented. In addition the clients must be updated with user and password prompting, and algorithms for verifying the logon routine. Another security issue is the transmission of information. As it is today, the XML-RPC calls containing the sensor information is unencrypted. The information is coded in XML, and is easy to interpret by humans. To prevent this from happening, the XML documents sent should be encrypted before transferred via XML-RPC. This could easily be done using existing encryption packages for Java, and few extra lines of code would have to be added.

To make the fish farm sensor information available on all computers on the Internet, a web interface could be useful. This could be implemented in HTML/JSP or AJAX using the already implemented RPC-handler class from the FiFaMoS Context Consumer.

References

[1] The Akogrimo project partners: Akogrimo:: Access to Knowledge Through the Grid in a Mobile World, (Akogrimo), *http://www.mobilegrids.org/* (last visited: 31.05.2006)

[2] Osland, P. O.,: *Final Integration Services Design and Implementation Report - Mobile Network Middleware Architecture, Design & Implementation* (Akogrimo), (2005)

[3] Osland, P. O., Viken, B., Solsvik, F., Nygreen, G., Wedvik, J., Myklebust, S. E., *Enabling context-aware applications,* (Telenor R&D/Gintel AS, Trondheim), (2005).

[4] Diaz Sendra, S: *Sea Cage Gateway – Fish Farm Control Station*, master thesis, (Telenor R&D/NTNU), (2006)

[5] Sospedra Cardona, R: *Sea Cage Gateway - Management System*, master thesis, Telenor R&D/NTNU (2006)

[6] Munch-Ellingsen, A: *A Container Based Middleware Approach Supporting Context-aware Service Development.* (University of Tromsø), (2005)

[7] Fleury, M., Lindfors, J.: Enabling Component Architectures with JMX, *http://www.onjava.com/pub/a/onjava/2001/02/01/jmx.html* (last visited: 10.06.2006)

[8] Word Reference dictionary: Definition of context, *http://www.wordreference.com/definition/context*, (last visited: 20.04.2006)

[9] Dey, A., Salber, D., Abowd, G.: *A context-based infrastructure for smart environments.* In: 1st International Workshop on Managing Interactions in Smart Environments (MANSE'99), (1999)

[10] Kouadri Mostéfaoui, S., Kouadri Mostéfaoui, G.: *Towards a contextualisation of service discovery and composition for pervasive environments.* Department of Computer Science, (University of Fribourg, Switzerland), (2003)

[11] Karen Henricksen, Jadwiga Indulska, and Andry Rakotonirainy: *Modelling Context Information in Pervasive Computing Systems.* School of Information Technology and Electrical Engineering (The University of Queensland), (2002)

[12] Jadwiga Indulska, Ricky Robinson, Andry Rakotonirainy, and Karen Henricksen: *Experiences in Using CC/PP in Context-Aware Systems.* School of Information Technology and Electrical Engineering (The University of Queensland), (2002)

[13] Li, Wei, *A Service Oriented SIP infrastructure for Adaptive and Context-Aware Wireless Services*, Department of Computer and Systems Sciences (Royal Institute of Technology), (2003)

[14] AKVAsmart ASA: AKVAsmart – We make your fish talk, (AKVAsmart), *http://www.akvasmart.no* (last visited: 28.05.2006)

[15] DSTC: Elvin notification messages, *http://elvin.dstc.com/intro/what.html* (last visited: 14.03.2006)

[16] Garmin Ltd: What is GPS?, (Garmin) *http://www.garmin.com/aboutGPS/* (last visited: 31.05.2006)

[17] GSM Association 2006: GPRS platform, (GSM World), *http://www.gsmworld.com/technology/gprs/index.shtml* (last visited: 31.05.2006)

[18] The 3rd Generation Partnership Project: About 3GPP, (3GPP), *http://www.3gpp.org/About/about.htm* (last visited: 20.04.2006)

[19] Apache Software Foundation: Apache XML-RPC, *http://ws.apache.org/xmlrpc/* (last visited: 06.06.2006)

[20] ObjectWeb/Enhydra: The home of kXML-RPC: *http://kxmlrpc.objectweb.org/* (last visited: 31.05.2006)

[21] Torres Serrano, J. U.: Comtor J2ME, *http://comtorj2me.sourceforge.net/* (last visited: 31.05.2006)

[22] CommLinx Solutions: NMEA examples, *http://www.commlinx.com.au/NMEA_sentences.htm* (last visited: 04.06.2006)

[23] USGS Eastern Region Geography: Universal Transverse Mercator (UTM) Grid, Fact Sheet, *http://erg.usgs.gov/isb/pubs/factsheets/fs07701.html* (last visited: 04.06.2006)

[24] KCS BV: TraceME track & trace modules, *http://www.traceme.tv/* (last visited: 31.05.2006)

[25] BlueTree Wireless data Inc.: BlueTree Wireless Modems, *http://www.bluetreewireless.com/products/wireless/details.asp?id=39* (last visited: 31.05.2006)

[26] Aplicom: Aplicom L-series - wireless GPRS/GSM device (M2M), *http://www.aplicom.com/lseries_m2m.html* (last visited: 31.05.2006)

[27] Teltonika: T-Box/GPS-307, *http://www.teltonika.lt/en/pages/view/?id=6* (last visited: 31.05.2006)

[28] Advantech Co. Ltd.: Advantech - Trusted ePlatform Services - ARK-3381 - Seven Serial Port Fanless Embedded Box Computer, *http://www.advantech.com/products/Model_Detail.asp?model_id=1-1TGX8Y&BU=ACG&PD=#* (last visited: 31.05.2006)

[29] Mort Bay Consulting: Jetty Java HTTP Servlet Server, *http://jetty.mortbay.org/jetty/index.html* (last visited: 20.04.2006)

[30] Apache Software Foundation: Apache Derby, *http://db.apache.org/derby/* (last visited: 05.04.2006)

[31] The Jabber Software Foundation: Jabber: Open Instant Messaging and a Whole Lot More, *http://www.jabber.org/* (last visited: 04.06.2006)

[32] The Codehaus: Drools home, *http://www.drools.org/* (last visited: 01.06.2006)

[33] Fowler, M., Scott, K.: UML Distilled Second Edition – *A brief guide to the standard object modelling language*. (2000).

[34] Martin, Robert C.: *UML for Java Programmers*. (Object Mentor Inc.), (2003).

[35] McLaughlin B.: *Java and XML*. (O'Reilly), (2000).

[36] St.Laurent, S., Johnston, J., Dumbill, E.: *Programming Web Services with XML-RPC*. (O'Reilly), (2001).

[37] Harold, Eliotte R., Scott Means, W.: *XML in a Nutshell – A Desktop Quick Reference*. (O'Reilly), (2002).

[38] Harold, Eliotte R.: Processing XML with Java – *A Guide to SAX, DOM, JDOM, JAXP, and TrAX*. (Addison-Wesley), (2003).

[39] Valacich Joseph S., George, Joey F., Hoffer, Jeffrey A.: *Essentials of Systems Analysis and Design*. (Prentice Hall), (2001).

[40] Winder, R., Roberts, G.: *Developing Java Software*. (Wiley), (2001).

[41] Orlowski, A.: AT&T WiMax trials, *http://www.theregister.co.uk/2005/10/27/wimax_world_att_trial/* (last visited: 06.06.2006)

[42] W3C: Extensible Markup Language (XML) 1.0 W3C recommendation, *http://www.w3.org/TR/REC-xml/* (last visited: 31.05.2005)

[43] Round Solutions: CAM-VGA100 User Manual, *http://www.roundsolutions.com/cmos-camera/CAM-VGA100-User-Manual%20V2-0.pdf* (last visited: 15.03.2006)

[44] Aplicom: A12 Imlet programming guide, *http://www.aplicom.com/documents/K502850_a12_imlet_programming_guide.pdf* (last visited: 20.04.2006)

[45] Aplicom: A12 Properties reference guide, *http://www.aplicom.com/documents/K502875_a12_properties_reference_guide.pdf (last visited: 20.04.2006)*

[46] Aplicom: A12 Software developers guide, *http://www.aplicom.com/documents/K502860_a12_sw_developers_guide.pdf* (last visited: 20.04.2006)

[47] Aplicom: A12 test board specification, *http://www.aplicom.com/documents/S100701_a12_test_board_spec.pdf* (last visited: 20.04.2006)

[48] Object Management Group, Inc: CORBA Frequently Asked Questions and resources, *http://www.omg.org/gettingstarted/corbafaq.htm* (last visited: 14.04.2006)

[49] Hao He: What Is Service-Oriented Architecture, *http://webservices.xml.com/pub/a/ws/2003/09/30/soa.html* (last visited: 31.05.2006)

[50] David Megginson: SAX, *http://www.saxproject.org* (last visited: 31.05.2006)

[51] Jonathan Stott: GPS format conversion *http://www.jstott.me.uk/jcoord/* (last visited: 20.04.2006)

[52] Website optimization: HTTP compression, *http://www.websiteoptimization.com/speed/tweak/compress/* (last visited: 31.05.2006)

Appendix A Description of the user interface

When implementing FiFaMoS, the goal were to make a system as near a finished product as possible. This led to that a lot of time was used to optimize the user experience, including graphical design, and GUI window and dialog design. Input verification and error handling have also been a subject of matter to make the demonstration as good as possible. In the end, the system works as intended, and not much implementation work remains before the product is good enough for the market. The main task of this appendix is to give an overview of the FiFaMoS system, and to present the final result.

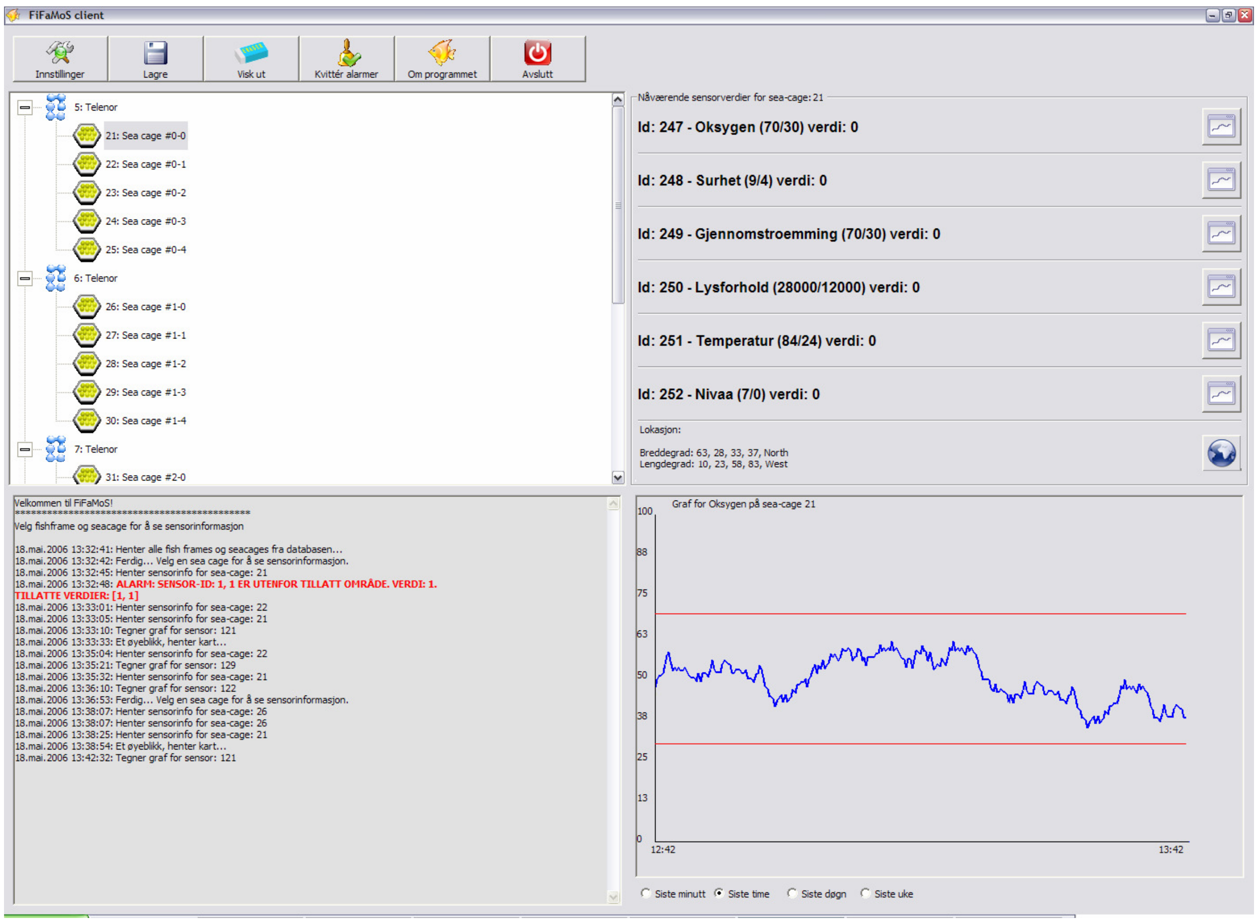
The system includes multiple clients:

- A Java ME mobile client for mobile phones and other handheld devices: FiFaMoS Mobile Context Consumer
- A Java SE application client for laptop/desktop system: FiFaMoS Context Consumer.

The user interfaces will be described in this appendix.

A.1 FiFaMoS Context Consumer

This client was implemented using Standard Widget Toolkit (SWT). This is an open-source framework for developing graphical user interfaces in Java. The main advantage of SWT is that the GUI will look and act the same as the operating system's native GUI.



Nåværende sensorverdier for sea-cage: 21

Id: 247 - Oksygen (70/30) verdi: 0

Id: 248 - Surhet (9/4) verdi: 0

Id: 249 - Gjennomstroemming (70/30) verdi: 0

Id: 250 - Lysforhold (28000/12000) verdi: 0

Id: 251 - Temperatur (84/24) verdi: 0

Id: 252 - Nivaa (7/0) verdi: 0

Lokasjon:

Breddegrad: 63, 28, 33, 37, North
Lengdegrad: 10, 23, 58, 83, West

```
Velkommen til FiFaMoS!
*********************************************
Velg fishframe og seacage for å se sensorinformasjon

18.mai.2006 13:32:41: Henter alle fish frames og seacages fra databasen...
18.mai.2006 13:32:42: Ferdig... Velg en sea cage for å se sensorinformasjon.
18.mai.2006 13:32:45: Henter sensorinfo for sea-cage: 21
18.mai.2006 13:32:48: ALARM: SENSOR-ID: 1, 1 ER UTENFOR TILLATT OMRÅDE. VERDI: 1.
TILLATTE VERDIER: [1, 1]
18.mai.2006 13:33:01: Henter sensorinfo for sea-cage: 22
18.mai.2006 13:33:05: Henter sensorinfo for sea-cage: 21
18.mai.2006 13:33:10: Tegner graf for sensor: 121
18.mai.2006 13:33:33: Et øyeblikk, henter kart...
18.mai.2006 13:35:04: Henter sensorinfo for sea-cage: 22
18.mai.2006 13:35:21: Tegner graf for sensor: 129
18.mai.2006 13:35:32: Henter sensorinfo for sea-cage: 21
18.mai.2006 13:36:10: Tegner graf for sensor: 122
18.mai.2006 13:36:53: Ferdig... Velg en sea cage for å se sensorinformasjon.
18.mai.2006 13:38:07: Henter sensorinfo for sea-cage: 26
18.mai.2006 13:38:07: Henter sensorinfo for sea-cage: 26
18.mai.2006 13:38:25: Henter sensorinfo for sea-cage: 21
18.mai.2006 13:38:54: Et øyeblikk, henter kart...
18.mai.2006 13:42:32: Tegner graf for sensor: 121
```

Figure 49: The graphical user interface of the FiFaMoS Context Consumer.

The main window is divided into four main parts: The list of frames and sea cages, the list of sensor values, a log section and a last part where graphs for historical sensor values can be shown.

The first part where sea cages can be chosen, is based on a two-level SWT tree widget. Fish frames are displayed in the first level and sea cages in the second. When a sea cage is chosen, the sensor information and GPS data from the sea cage is displayed in section top right. This information is continually updated, and buttons for displaying graphs and maps are in this section as well. Pressing the map button opens a new dialog showing the location of the sea cage in a map.

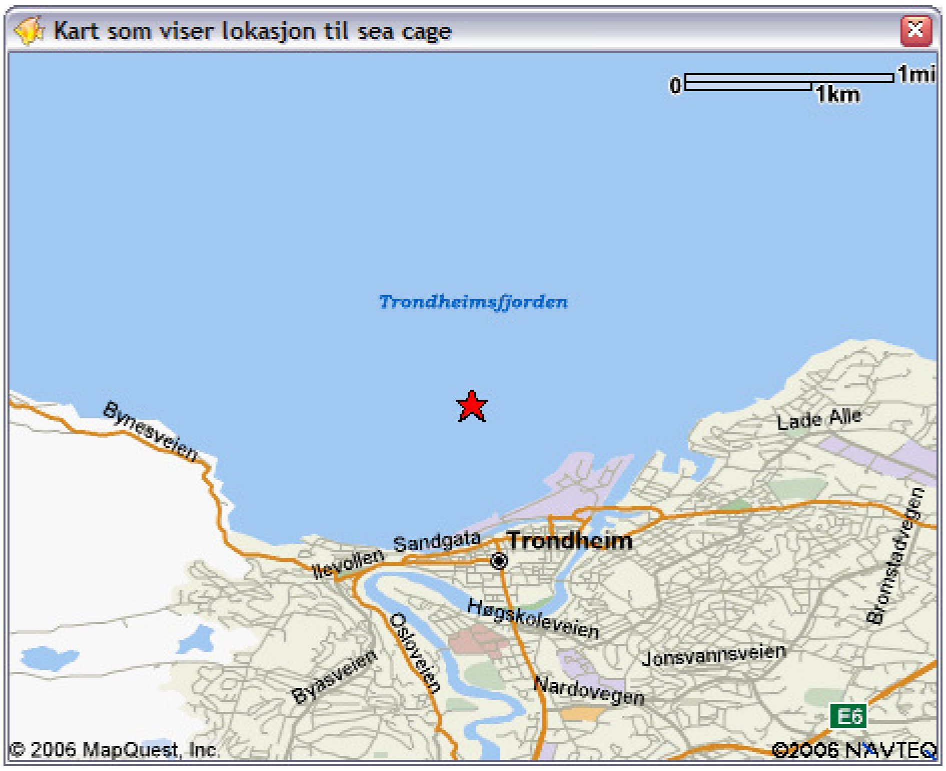
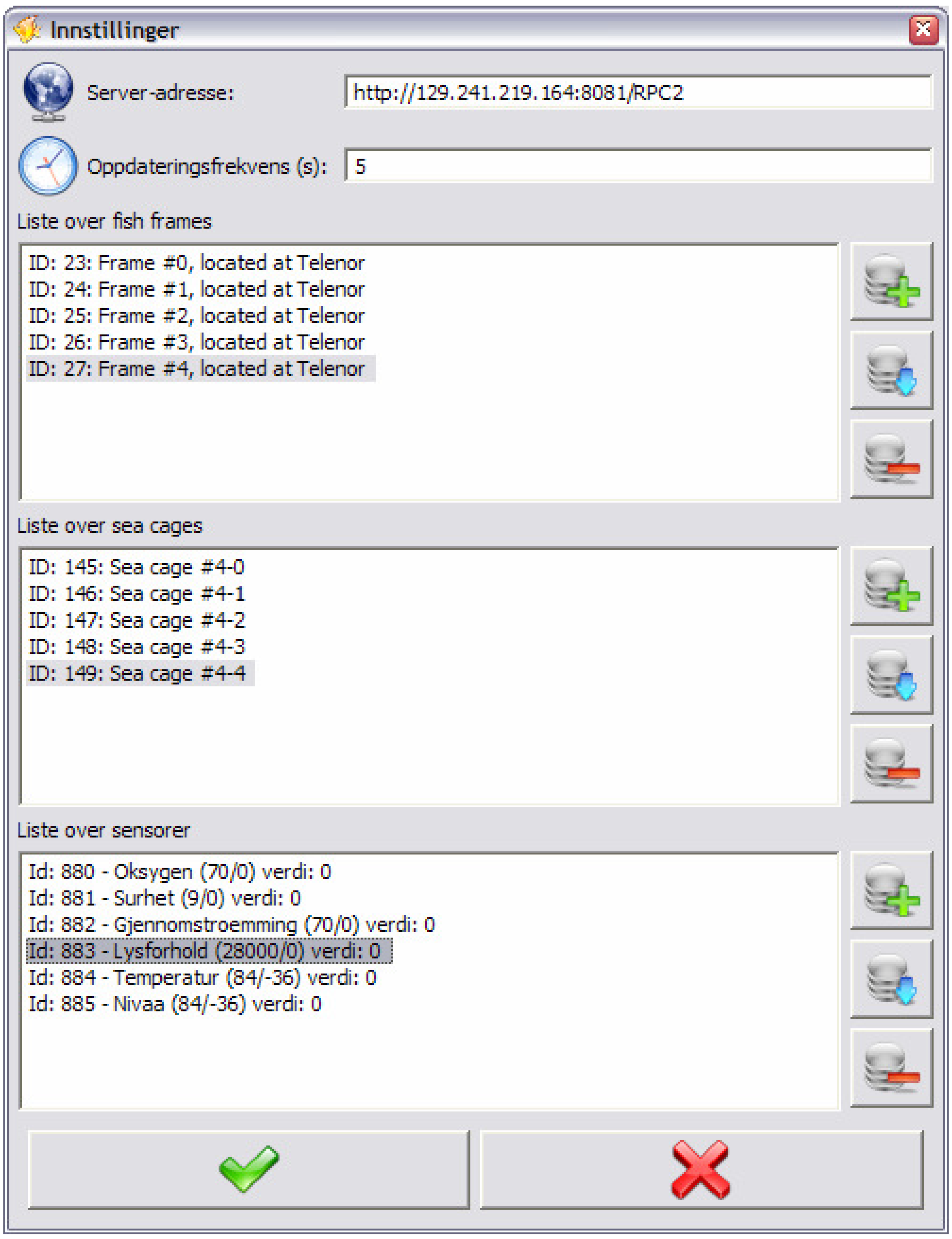


Figure 50: Map showing the sea cage location (the red star).

To display historical sensor values, a graph is used. This graph can display sensor values from preset periods. The period is chosen by radio buttons under the graph, and the graph itself will be displayed bottom right. The graph is drawn in X and Y coordinates in a canvas, and it adapts itself to the window size. Every command that is made to the system is logged with a timestamp in the logging section. Alarms and abnormal sensor values will be displayed here as well. The user can either save the log to a file on the hard drive, or choose to blank the logging section. The widget used is StyledText which makes is possible to use various font types and colours. Alarms will be displayed in bold red text for the operator's attention.

The settings dialog

When pressing the settings button in the main window, a new dialog appears. Here it is possible to alter various parameters of the system. These parameters include the context manager server address, how often sensor values should be refreshed, and the opportunity to configure the fish farm.


Innstillinger

Server-adresse: http://129.241.219.164:8081/RPC2

Oppdateringsfrekvens (s): 5

Liste over fish frames

ID: 23: Frame #0, located at Telenor
ID: 24: Frame #1, located at Telenor
ID: 25: Frame #2, located at Telenor
ID: 26: Frame #3, located at Telenor
ID: 27: Frame #4, located at Telenor

Liste over sea cages

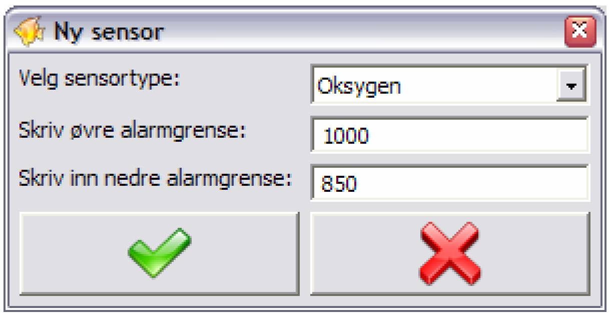
ID: 145: Sea cage #4-0
ID: 146: Sea cage #4-1
ID: 147: Sea cage #4-2
ID: 148: Sea cage #4-3
ID: 149: Sea cage #4-4

Liste over sensorer

Id: 880 - Oksygen (70/0) verdi: 0
Id: 881 - Surhet (9/0) verdi: 0
Id: 882 - Gjennomstroemming (70/0) verdi: 0
Id: 883 - Lysforhold (28000/0) verdi: 0
Id: 884 - Temperatur (84/-36) verdi: 0
Id: 885 - Nivaa (84/-36) verdi: 0

Figure 51: Screenshot of the settings dialog

The fish farm configuration option makes it possible to add, remove or edit frames, sea cages and sensors. The frames, sea cages and sensors are displayed in tables, where each table has buttons for adding, removing or editing the frame, sea cage or sensor. Pressing the add or edit button triggers a new dialog where the parameters can be set or edited.

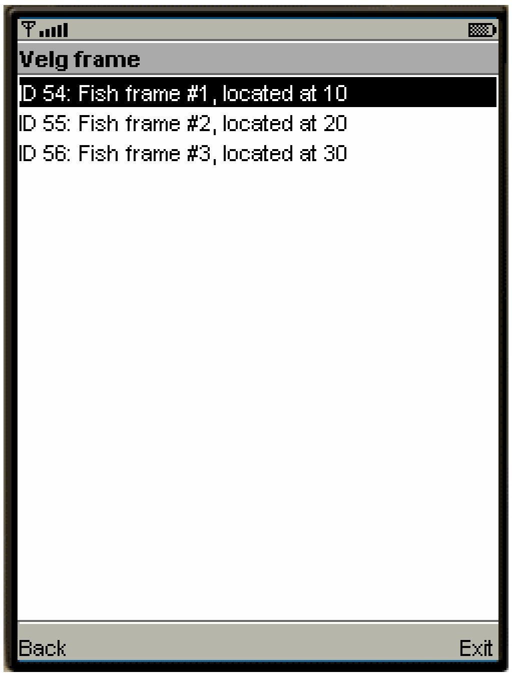

Ny sensor

Velg sensortype:	Oksygen
Skriv øvre alarmgrense:	1000
Skriv inn nedre alarmgrense:	850

Figure 52: The dialog for adding a new sensor

A.2 FiFaMoS Mobile Context Consumer

To navigate between frames, sea cages and sensors in the FiFaMoS Mobile Context Consumer, lists are used. The first list displayed is the frame list. When a frame is chosen, a list over its sea cages is displayed. The same way, the sensor values of a sea cage are displayed when a sea cage is chosen.


Velg frame

ID 54: Fish frame #1, located at 10
ID 55: Fish frame #2, located at 20
ID 56: Fish frame #3, located at 30

Figure 53: List over the fish frames from the FiFaMoS Mobile Context Consumer

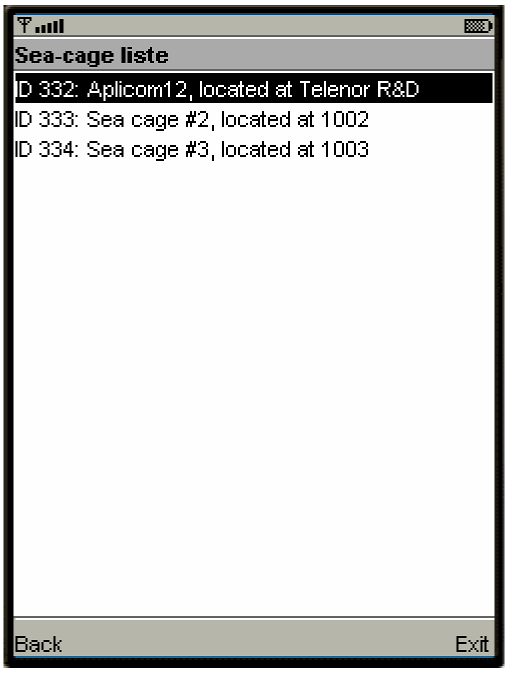

Sea-cage liste
ID 332: Aplicom12, located at Telenor R&D
ID 333: Sea cage #2, located at 1002
ID 334: Sea cage #3, located at 1003

Figure 54: List over the sea cages from the FiFaMoS Mobile Context Consumer

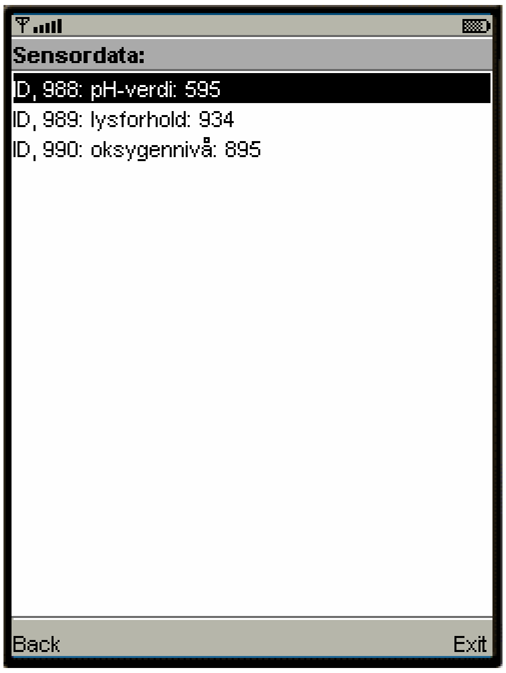

Sensordata:
ID, 988: pH-verdi: 595
ID, 989: lysforhold: 934
ID, 990: oksygennivå: 895

Figure 55: The list of the current sensor values of a sea cage from the FiFaMoS Mobile Context Consumer.

By choosing one of the sensors in the sensor list, historical values can be shown. Before the values are shown, the time span have to be chosen. This is done in a list, where “last hour”, "last 24 hours", "last week" and "last month" are options. When the time span is chosen, the historical sensor values are presented in a graph like shown in Figure 56.

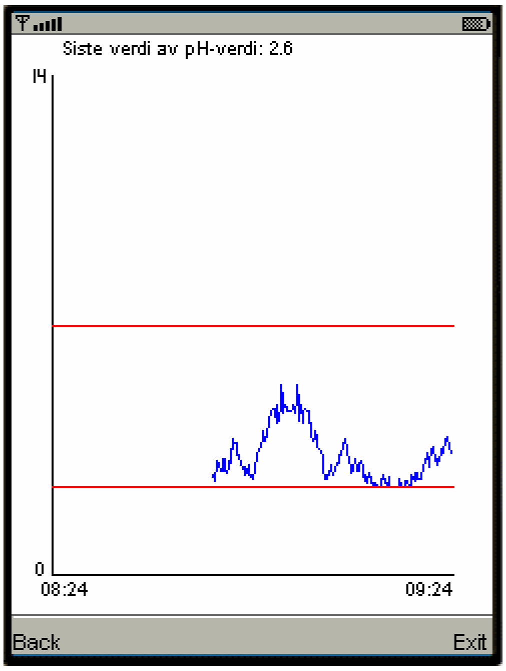
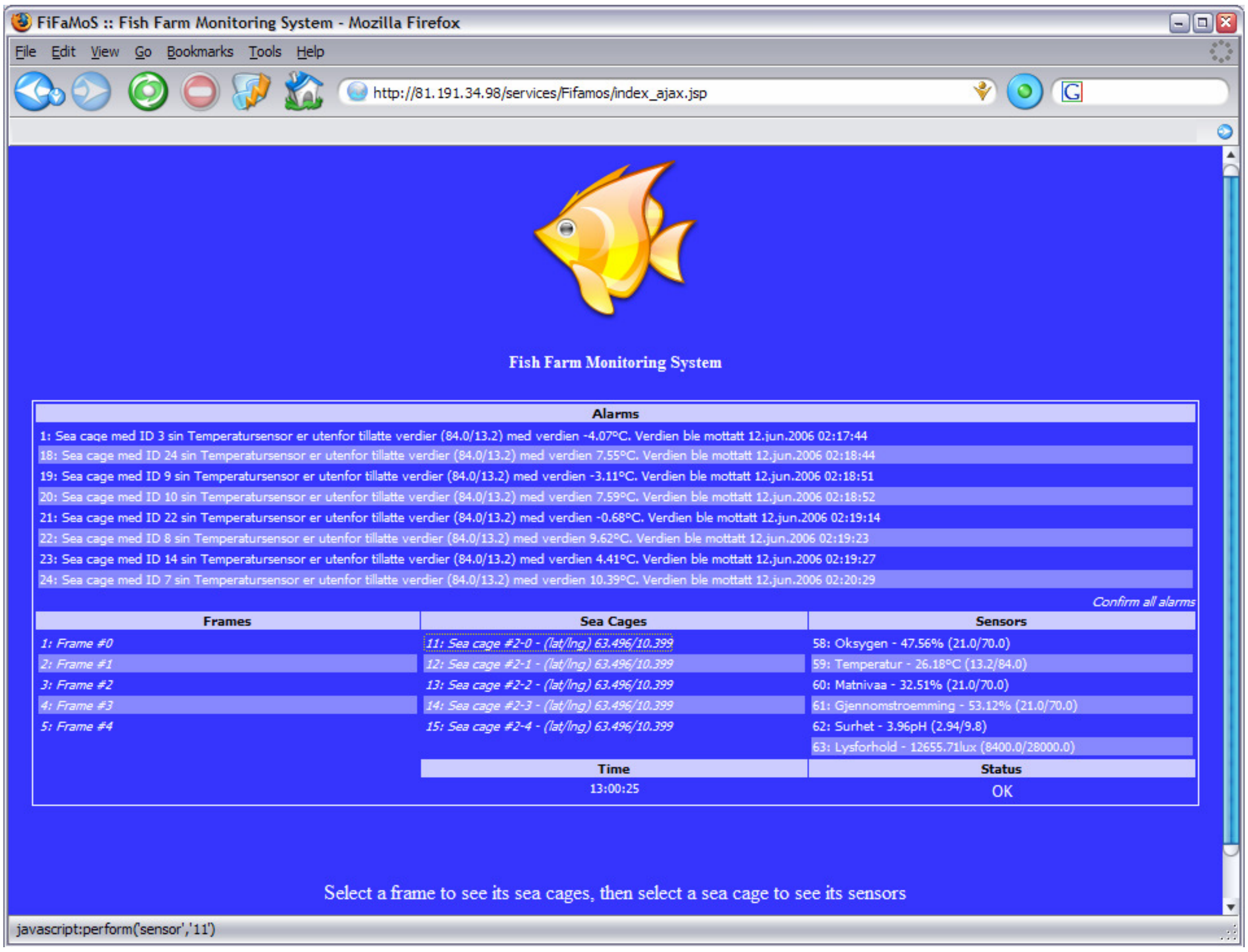

Siste verdi av pH-verdi: 2.6

Figure 56: The graph showing historical sensor values from the FiFaMoS Mobile Context Consumer

In addition the application has a start menu the settings windows can be accessed from, other choices in the start menu includes starting the surveillance, or exit the program.

A.3 FiFaMoS web interface

During the last hours of the project, a web interface was developed. This is based on AJAX (Asynchronous Javascript and XML) technology, which enables dynamic updating of the web site. In addition JSP (Java Server Pages) is used for generating the XML of the data from the context manager.The web interface is located at at: *http://'context manager ip':'port'/services/Fifamos/* The interface displays both sensor information and alarms, and looks like this:


FiFaMoS :: Fish Farm Monitoring System - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://81.191.34.98/services/Fifamos/index_ajax.jsp
Fish Farm Monitoring System
Confirm all alarms
javascript:perform('sensor','11')

Alarms
1: Sea cage med ID 3 sin Temperatursensor er utenfor tillatte verdier (84.0/13.2) med verdien -4.07°C. Verdien ble mottatt 12.jun.2006 02:17:44
18: Sea cage med ID 24 sin Temperatursensor er utenfor tillatte verdier (84.0/13.2) med verdien 7.55°C. Verdien ble mottatt 12.jun.2006 02:18:44
19: Sea cage med ID 9 sin Temperatursensor er utenfor tillatte verdier (84.0/13.2) med verdien -3.11°C. Verdien ble mottatt 12.jun.2006 02:18:51
20: Sea cage med ID 10 sin Temperatursensor er utenfor tillatte verdier (84.0/13.2) med verdien 7.59°C. Verdien ble mottatt 12.jun.2006 02:18:52
21: Sea cage med ID 22 sin Temperatursensor er utenfor tillatte verdier (84.0/13.2) med verdien -0.68°C. Verdien ble mottatt 12.jun.2006 02:19:14
22: Sea cage med ID 8 sin Temperatursensor er utenfor tillatte verdier (84.0/13.2) med verdien 9.62°C. Verdien ble mottatt 12.jun.2006 02:19:23
23: Sea cage med ID 14 sin Temperatursensor er utenfor tillatte verdier (84.0/13.2) med verdien 4.41°C. Verdien ble mottatt 12.jun.2006 02:19:27
24: Sea cage med ID 7 sin Temperatursensor er utenfor tillatte verdier (84.0/13.2) med verdien 10.39°C. Verdien ble mottatt 12.jun.2006 02:20:29

Frames	Sea Cages	Sensors
1: Frame #0	11: Sea cage #2-0 - (lat/lng) 63.496/10.399	58: Oksygen - 47.56% (21.0/70.0)
2: Frame #1	12: Sea cage #2-1 - (lat/lng) 63.496/10.399	59: Temperatur - 26.18°C (13.2/84.0)
3: Frame #2	13: Sea cage #2-2 - (lat/lng) 63.496/10.399	60: Matnivaa - 32.51% (21.0/70.0)
4: Frame #3	14: Sea cage #2-3 - (lat/lng) 63.496/10.399	61: Gjennomstroemming - 53.12% (21.0/70.0)
5: Frame #4	15: Sea cage #2-4 - (lat/lng) 63.496/10.399	62: Surhet - 3.96pH (2.94/9.8)
		63: Lysforhold - 12655.71lux (8400.0/28000.0)

Time	Status
13:00:25	OK

Select a frame to see its sea cages, then select a sea cage to see its sensors

Figure 57: The FiFaMoS web interface

Appendix B   Code listing

B.1 Input component meta file

The input component meta file defines the component type, binding type used, and the persistence model. This is described in an XML-file.

```
<?xml version = "1.0" encoding="UTF-8" standalone = "no"?>
<!DOCTYPE METAINFORMATION SYSTEM "./config/InputMeta_DTD.dtd">
<METAINFORMATION>
  <component>
    <version>0.5</version>
    <type>input</type>
    <description>Input component for Fifamos</description>
    <bindingtype>xmlrpc</bindingtype>

    <!-- All tables have the primary key field 'id' automatically set -->

    <pmodel>
      <table>
        <tablename>frame</tablename>
        <field>
          <fieldname>location</fieldname>
          <fieldtype>string</fieldtype>
        </field>
        <field>
          <fieldname>description</fieldname>
          <fieldtype>string</fieldtype>
        </field>
        <field>
          <fieldname>mobile</fieldname>
          <fieldtype>string</fieldtype>
        </field>
        <field>
          <fieldname>email</fieldname>
          <fieldtype>string</fieldtype>
        </field>
        <field>
          <fieldname>updated</fieldname>
          <fieldtype>int</fieldtype>
        </field>
      </table>
      <table>
        <tablename>seacage</tablename>
        <field>
          <fieldname>latitude</fieldname>
          <fieldtype>string</fieldtype>
        </field>
        <field>
          <fieldname>longitude</fieldname>
          <fieldtype>string</fieldtype>
        </field>
        <field>
          <fieldname>initialLatitude</fieldname>
          <fieldtype>string</fieldtype>
        </field>
        <field>
          <fieldname>initialLongitude</fieldname>
          <fieldtype>string</fieldtype>
        </field>
        <field>
          <fieldname>description</fieldname>
          <fieldtype>string</fieldtype>
        </field>
        <field>
          <fieldname>imei</fieldname>
          <fieldtype>string</fieldtype>
        </field>
        <field>
          <fieldname>interval</fieldname>
          <fieldtype>int</fieldtype>
        </field>
        <field>
          <fieldname>frameid</fieldname>
          <fieldtype>int</fieldtype>
        </field>
      </table>
      <table>
        <tablename>sensor</tablename>
        <field>
          <fieldname>sensortypeid</fieldname>
          <fieldtype>int</fieldtype>
        </field>
        <field>
          <fieldname>maxvalue</fieldname>
          <fieldtype>double</fieldtype>
        </field>
        <field>
          <fieldname>minvalue</fieldname>
          <fieldtype>double</fieldtype>
        </field>
        <field>
          <fieldname>seacageid</fieldname>
          <fieldtype>int</fieldtype>
        </field>
      </table>
      <table>
        <tablename>sensorvalue</tablename>
        <field>
          <fieldname>value</fieldname>
          <fieldtype>double</fieldtype>
        </field>
        <field>
          <fieldname>received</fieldname>
          <fieldtype>timestamp</fieldtype>
        </field>
        <field>
```

```
        <fieldname>sensorid</fieldname>
        <fieldtype>int</fieldtype>
      </field>
    </table>
    <table>
      <tablename>alarm</tablename>
      <field>
        <fieldname>seacageid</fieldname>
        <fieldtype>int</fieldtype>
      </field>
      <field>
        <fieldname>sensorid</fieldname>
        <fieldtype>string</fieldtype>
      </field>
      <field>
        <fieldname>message</fieldname>
        <fieldtype>string</fieldtype>
      </field>
      <field>
        <fieldname>issent</fieldname>
        <fieldtype>int </fieldtype>
      </field>
      <field>
        <fieldname>confirmed</fieldname>
        <fieldtype>timestamp</fieldtype>
      </field>
    </table>
    <table>
      <tablename>sensortype</tablename>
      <field>
        <fieldname>type</fieldname>
        <fieldtype>string</fieldtype>
      </field>
      <field>
        <fieldname>description</fieldname>
        <fieldtype>string</fieldtype>
      </field>
      <field>
        <fieldname>unit</fieldname>
        <fieldtype>string</fieldtype>
      </field>
      <field>
        <fieldname>maxrange</fieldname>
        <fieldtype>double</fieldtype>
      </field>
      <field>
        <fieldname>minrange</fieldname>
        <fieldtype>double</fieldtype>
      </field>
    </table>
   </pmodel>
  </component>
</METAINFORMATION>
```

Code 38: The input component meta file

B.2 Output component meta file

The output component meta file describes the component type.

```
<?xml version = "1.0" encoding="UTF-8" standalone = "no"?>
<!DOCTYPE METAINFORMATION SYSTEM "./config/OutputMeta_DTD.dtd">
<METAINFORMATION>
  <component>
    <version>0.5</version>
    <type>output</type>
    <description>Output component for Fifamos</description>
  </component>
</METAINFORMATION>
```
Code 39: The output component meta file

B.3 Deployment descriptor

The deployment descriptor is used to define the service and its components.

```
<?xml version = "1.0" encoding="UTF-8" standalone = "no"?>
<!DOCTYPE METAINFORMATION SYSTEM "./config/DeploymentDescriptor_DTD.dtd">
<METAINFORMATION>
  <service>
    <name>Fifamos</name>
    <version>0.5</version>
    <description>Beskrivelse av Fifamos</description>
    <dbType>Derby</dbType>
  </service>
  <deploy>
    <component>
      <name>Input</name>
      <class>com.telenor.apms.fifamos.Input</class>
      <metafile>InputMeta.xml</metafile>
    </component>
    <component>
      <name>Output</name>
      <class>com.telenor.apms.fifamos.Output</class>
      <metafile>OutputMeta.xml</metafile>
    </component>
  </deploy>
</METAINFORMATION>
```
Code 40: Deployment descriptor

B.4 XML representations

Most of the objects that are transferred between the nodes in the system are represented as an XML string. This appendix describes the configuration of these XML representations.

```
<XML>
  <frame>
    <id> 1 </id>
    <location> location </location>
    <description> description </description>
    <mobile> 99989796 </mobile>
    <email> operator@fifamos.com </email>
  </frame>
</XML>
```

Code 41: XML representation of a fish frame

```
<XML>
  <seacage>
    <id>  2  </id>
    <latitude> 63.23112 </latitude>
    <longitude> 10.34267 </longitude>
    <description> description </description>
    <imei> 352540000046895 </imei>
    <interval> 10 </interval>
    <frameid> 1 </frameid>
  </seacage>
</XML>
```

Code 42: XML representation of a sea cage

```
<XML>
  <sensor>
    <id> 1 </id>
    <sensortypeid> 1 </sensortypeid>
    <maxrange> 120 </maxrange>
    <minrange> -40 </minrange>
    <maxvalue> 25 </maxvalue>
    <minvalue> 4 </minvalue>
    <seacageid> 2 </seacageid>
  </sensor>
</XML>
```

Code 43: XML representation of a sensor

```
<xml>
  <sensorvalue>
    <sensorid>3</sensorid>
    <value>253</value>
  </sensorvalue>
</xml>
```

Code 44: XML representation of a sensor value

```
<xml>
  <command type=”add”>
    <frame>
      <id>0</id>
      <location>Trondheimsfjorden</location >
      <description>Testframe</description>
      <mobile>99989796</mobile>
      <email>operator@fifamos.net</email>
    </frame>
  </command>
</xml>
```
Code 45: XML representation of a command for adding a frame

B.5 Class for reading GPS data

The GPSControl is used as a wrapper for the GpsModule class available in the Aplicom N12i. It is located in the package com.telenor.apms.fifamos.n12.utils.

```
package com.telenor.apms.fifamos.n12.utils;

import com.nokia.m2m.orb.idl.gps.GpsData;
import com.nokia.m2m.orb.idl.gps.GpsModule;
import com.nokia.m2m.orb.idl.gps.GpsModuleHelper;
import com.nokia.m2m.orb.idl.gps.NoGpsDataAvailableException;

public class GPSControl {
  private GpsModule gpsModule;
  private GpsData data;

  private long latdeg;
  private long latmin;
  private long latsec;
  private long latmil;
  private long lngdeg;
  private long lngmin;
  private long lngsec;
  private long lngmil;

  public GPSControl(org.omg.CORBA.ORB orb) {

//   Get refrence to the GPS module.
    String url =
      "corbaloc::127.0.0.1:19740/ORB/OA/IDL:gps/GpsModule:1.0";
    org.omg.CORBA.Object ref = orb.string_to_object(url);

//   get the GPS object
    gpsModule = GpsModuleHelper.narrow(ref);
  }

  /** Method for reading the GPS position.
   * This method must be called at least once before calling
   * the different get-methods.
   *
   * @throws NoGpsDataAvailableException
   */
  public void read() throws NoGpsDataAvailableException {
    data = gpsModule.getGpsData();

    latdeg = data.position.lat.degrees;
    latmin = data.position.lat.minutes;
    latsec = data.position.lat.seconds;
    latmil = data.position.lat.milliseconds;

    lngdeg = data.position.lon.degrees;
    lngmin = data.position.lon.minutes;
    lngsec = data.position.lon.seconds;
    lngmil = data.position.lon.milliseconds;
  }
  /**
   * @return the latitude degrees
   */
  public long getLatdeg() {
   return latdeg;
  }

  /**
   * @return the latitude minutes
   */
  public long getLatmin() {
   return latmin;
  }

  /**
   * @return the latitude seconds
   */
  public long getLatsec() {
   return latsec;
  }

  /**
   * @return the latitude milliseconds
   */
  public long getLatmil() {
   return latmil;
  }

  /**
   * @return the longitude degrees
   */
  public long getLngdeg() {
   return lngdeg;
  }

  /**
   * @return the longitude minutes
   */
  public long getLngmin() {
   return lngmin;
  }

  /**
   * @return the longitude seconds
   */
  public long getLngsec() {
   return lngsec;
  }

  /**
   * @return the longitude milliseconds
   */
  public long getLngmil() {
   return lngmil;
  }
}
```

Code 46: The GPSControl class, used as a wrapper for adding GPS data

B.6 Class for updating the database

Since the database abstraction on the APMS context manager did not support the update-operation, an extra class had to be implemented. This class was put on the context manager along with the insert, select and delete classes.

```
package coms.tools.relational;

import java.sql.SQLException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.Set;
import javax.management.MBeanServer;
import javax.management.ObjectName;
import coms.CoMS;
import coms.database.ConnectionException;
import coms.database.DatabaseConnection;

/**
 * Insert is used to update data in a table in a relational database.
 * Based on coms.tools.relational.Insert.java
 *
 */
public class Update {
 private String prevCol = null;

 private Object prevVal = null;

 private HashMap<String, Object> data = new HashMap<String, Object>();

 private String tbl;

 private String dbType;

 private String conditionCol;

 private Object conditionVal;

 public Update(String tableName, String DBType) {
  this.tbl = tableName;
  this.dbType = DBType;
 }

 /**
  * Inserts the value into the column
  *
  * @param col
  *            column where the value is to be placed
  * @param value
  *            to be put into the database
  */
 public void column(String col, Object value) {
  prevCol = col;
  prevVal = data.get(col);
  data.put(col, value);
 }

 public void condition(String col, Object value) {
  conditionCol = col;
  conditionVal = value;
 }

 /**
  * Removes the last value to be put into the database
  */
 public void undo() {
  if (prevVal == null)
   data.remove(prevCol);
  else
   data.put(prevCol, prevVal);
 }

 /**
  * Will put the previously specified data into the database.
  *
  * @param tbl
  *            The table the data will be inserted into
  * @param dbType
  *            The type of the database, can be found in
  *            "getMetaData().getDbType()
  * @throws SQLException
  *             if there is something wrong in the generated SQL string
  * @throws ConnectionException
  *             if a connection cannot be aquired
  */
 public void execute() throws SQLException, ConnectionException {
  Iterator it = data.keySet().iterator();
  String sqlStr = "UPDATE " + tbl + " SET ";
  // Iterates over the data picking out the fields in each row
  while (it.hasNext()) {
   // Drops the commas the last time
   sqlStr += it.next() + " = ?" + (it.hasNext() ? "," : "");

  }
  sqlStr += " WHERE " + conditionCol + " = " + conditionVal;

  System.out.println("Update SQL string: " + sqlStr);

  try {
   DatabaseConnection con = getDatabaseConnection(dbType);
   con.executeSQL(sqlStr, data.values().toArray(), false);
   con.release();
  } catch (Exception e) {
   e.printStackTrace();
  }
  // Sends an event that there has been done changes to table
  notifyEvent(tbl);
 }

 /**
  * Puts a table containing values into a database
  *
  * @param tbl
  *            The table to be put into the database
  * @param dbType
  *            The type of the database, can be found in
  *            "getMetaData().getDbType()
  * @param data
  *            Component that called this method
  * @throws SQLException
  *             if there is something wrong in the generated SQL string
  * @throws ConnectionException
  *             if a connection cannot be aquired
  */
 public static void execute(String tbl, String dbType,
   Set<Map<String, Object>> data, String condCol, String condVal)
   throws SQLException, ConnectionException {
  ArrayList<Object> val = new ArrayList<Object>();
  Map row;
  String fieldName;
  String sqlStr = "UPDATE " + tbl + " SET ";
  Iterator field;
  Iterator it = data.iterator();
  // Iterates over the data picking out the fields in each row
  while (it.hasNext()) {
   row = (Map) it.next();
   field = row.keySet().iterator();
   while (field.hasNext()) {
    // Drops the commas the first time

    fieldName = (String) field.next();
    sqlStr += fieldName + " = ?" + (it.hasNext() ? "," : "");
    val.add(row.get(fieldName));
   }
  }

  sqlStr += " WHERE " + condCol + " = " + condVal + ")";

  try {
   DatabaseConnection con = getDatabaseConnection(dbType);
   con.executeSQL(sqlStr, val.toArray(), false);
   con.release();
  } catch (Exception e) {
   e.printStackTrace();
  }
  // Sends an event that there has been done changes to table
  notifyEvent(tbl);
 }

 private static DatabaseConnection getDatabaseConnection(String type)
   throws Exception {
  MBeanServer server = CoMS.getServerHandle();
  ObjectName db = new ObjectName(server.getDefaultDomain() + ":Name="
    + type + ", Type=Core");
  return (DatabaseConnection) server.getAttribute(db, "Connection");
 }

 /**
  * Notifies all components of a table change using the EventHandler.
  *
```

```
   * @param String
   *            table
   */
  private static void notifyEvent(String table) throws ConnectionException {
    MBeanServer server = CoMS.getServerHandle();
    try {
      // If a server is found, choose which type of database we want to
      // contact
      if (server != null) {
        ObjectName componentAgent = new ObjectName(server
            .getDefaultDomain()
            + ":Name=EventHandler, Type=Core");
        server.invoke(componentAgent, "notifyEvent",
            new Object[] { table }, new String[] { String.class
                .getName() });
      } else
        throw new ConnectionException("Could not find mbeanserver");
    } catch (Exception e) {
      // This should not occur
      e.printStackTrace();
    }
  }
}
```

Code 47: The database update class

B.7 The sensor simulator

```
package com.telenor.apms.fifamos.utils;

import java.io.ByteArrayInputStream;
import java.net.URL;
import java.util.Vector;

import org.apache.xmlrpc.XmlRpcClient;
import org.xml.sax.Attributes;
import org.xml.sax.InputSource;
import org.xml.sax.SAXException;
import org.xml.sax.XMLReader;
import org.xml.sax.helpers.DefaultHandler;
import org.xml.sax.helpers.XMLReaderFactory;


public class SensorSimClass extends DefaultHandler {
// private static final String _URL = "http://129.241.219.164:8081/RPC2";
 private static final String _URL = "http://localhost:8080/RPC2";
// private static final String _URL = "http://81.191.34.98:80/RPC2";
 private String imei;
 private int frameid;
 private int seacageid;
 private int index;
 private int[] sensorid;
 private StringBuffer xml = new StringBuffer();
 private XmlRpcClient xmlrpc;
 private Vector<String> params;

 public SensorSimClass(int imei, int sensors) {
  try {
   this.imei = String.valueOf(imei);
   sensorid = new int[sensors];
   URL url = new URL(_URL);
   xmlrpc = new XmlRpcClient(url);
   params = new Vector<String>();

   params.addElement(this.imei);

   String xmlconfig = (String) xmlrpc.execute("Output.getConfigAsXml",
params);

   XMLReader parser = XMLReaderFactory.createXMLReader();
   parser.setContentHandler(this);
   parser.parse(new InputSource(new
ByteArrayInputStream(xmlconfig.getBytes())));

  } catch (Exception e) {
   e.printStackTrace();
  }
 }

 public void pulse() {
  try {
    params.removeAllElements();

      xml.delete(0, xml.length());

      xml.append("<xml>");
//      xml.append("<seacageid>"+Config.seacageid+"</seacageid>");
      for(int i=0; i<sensorid.length;i++) {
        // creating xml elements
        if(sensorid[i]!=-1) {
          xml.append("<command type=\"add\">\n");
          xml.append("<sensorvalue>\n");
          xml.append("<sensorid>"+sensorid[i]+"</sensorid>\n");
//          xml.append("<value>"+(int)(Math.random()*2800)+"</value>\n");
          xml.append("<value>"+ ((Math.random()*1000) + 600) +"</value>\n");
          xml.append("</sensorvalue>\n");
          xml.append("</command>\n");
        }
      }
      xml.append("<command type=\"update\">\n");
      xml.append(
          "<gps>" +
          "<seacageid>" + seacageid + "</seacageid>" +
          "<latlng " +
            "lat=\""+ 63 + " " + 29 + " " + 47 + "\" " +
            "lng=\""+ 10 + " " + 23 + " " + 57 + "\" " +
          "/>" +
          "</gps>");
      xml.append("</command>\n");
      xml.append("</xml>");

      params.add(xml.toString());

      xmlrpc.execute("Input.postSourceXml", params);
    } catch (Exception e) {
      e.printStackTrace();
    }
  }

  @Override
  public void startDocument() throws SAXException {
  }

  @Override
  public void startElement(String uri, String localName, String qName,
Attributes attributes) throws SAXException {
    if(qName.equals("frame"))
      frameid = Integer.parseInt(attributes.getValue("value"));
    else if(qName.equals("seacage"))
      seacageid =  Integer.parseInt(attributes.getValue("value"));
    else if(qName.equals("sensorid"))
      sensorid[index++] = Integer.parseInt(attributes.getValue("value"));

  }

  @Override
  public void characters(char[] chars, int start, int length) throws
SAXException {
  }

  @Override
  public void endElement(String uri, String localName, String qName) throws
SAXException {

  }

  @Override
  public void endDocument() throws SAXException {
  }
}
```

Code 48: The sensor simulator source code

B.8 Bug fix: APMS Derby.java file

After installing a new version of the APMS context manager, the system did not work. It was impossible to create tables. The following stack trace was shown in the console:

```
javax.management.MBeanException: Exception thrown in operation createTable
        at
com.sun.jmx.mbeanserver.StandardMetaDataImpl.invoke(StandardMetaDataI
mpl.java:435)
        at
com.sun.jmx.mbeanserver.MetaDataImpl.invoke(MetaDataImpl.java:220)
        at
com.sun.jmx.interceptor.DefaultMBeanServerInterceptor.invoke(DefaultM
BeanServerInterceptor.java:815)
        at
com.sun.jmx.mbeanserver.JmxMBeanServer.invoke(JmxMBeanServer.java:784 )
        at coms.mbeans.Deployer.createTable(Deployer.java:755)
        at coms.mbeans.Deployer.createTables(Deployer.java:827)
        at coms.mbeans.Deployer.deploy(Deployer.java:856)
        at coms.mbeans.Deployer.deployComponents(Deployer.java:455)
        at coms.mbeans.Deployer.deployLoadedJars(Deployer.java:369)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at
sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.
java:39)
        at
sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAcces
sorImpl.java:25)
        at java.lang.reflect.Method.invoke(Method.java:585)
        at
com.sun.jmx.mbeanserver.StandardMetaDataImpl.invoke(StandardMetaDataI
mpl.java:414)
        at
com.sun.jmx.mbeanserver.MetaDataImpl.invoke(MetaDataImpl.java:220)
        at
com.sun.jmx.interceptor.DefaultMBeanServerInterceptor.invoke(DefaultM
BeanServerInterceptor.java:815)
        at
com.sun.jmx.mbeanserver.JmxMBeanServer.invoke(JmxMBeanServer.java:784 )
        at
coms.mbeans.ComponentLoader.deployLoadedJars(ComponentLoader.java:127 )
        at
coms.mbeans.ComponentLoader.deployJarFile(ComponentLoader.java:71)
        at coms.mbeans.ComponentLoader.checkFile(ComponentLoader.java:57)
        at
coms.mbeans.ComponentLoader.checkDeployFolder(ComponentLoader.java:15 8)
        at
coms.mbeans.ComponentLoader.loadComponents(ComponentLoader.java:138)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at
sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.
java:39)
        at
sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAcces
sorImpl.java:25)
```

```
        at java.lang.reflect.Method.invoke(Method.java:585)
        at
com.sun.jmx.mbeanserver.StandardMetaDataImpl.invoke(StandardMetaDataI
mpl.java:414)
        at
com.sun.jmx.mbeanserver.MetaDataImpl.invoke(MetaDataImpl.java:220)
        at
com.sun.jmx.interceptor.DefaultMBeanServerInterceptor.invoke(DefaultM
BeanServerInterceptor.java:815)
        at
com.sun.jmx.mbeanserver.JmxMBeanServer.invoke(JmxMBeanServer.java:784 )
        at coms.CoMS.loadComponents(CoMS.java:85)
        at coms.CoMS.<init>(CoMS.java:355)
        at coms.CoMS.main(CoMS.java:40)
Caused by: SQL Exception: Syntax error: Encountered "," at line 1, column
111.
        at org.apache.derby.impl.jdbc.Util.generateCsSQLException(Unknown
Source )
        at
org.apache.derby.impl.jdbc.TransactionResourceImpl.wrapInSQLException
(Unknown Source)
        at
org.apache.derby.impl.jdbc.TransactionResourceImpl.handleException(Un known
Source)
        at
org.apache.derby.impl.jdbc.EmbedConnection.handleException(Unknown So urce)
        at
org.apache.derby.impl.jdbc.ConnectionChild.handleException(Unknown So urce)
        at org.apache.derby.impl.jdbc.EmbedPreparedStatement.<init>(Unknown
Sour ce)
        at
org.apache.derby.impl.jdbc.EmbedPreparedStatement20.<init>(Unknown So urce)
        at
org.apache.derby.impl.jdbc.EmbedPreparedStatement30.<init>(Unknown So urce)
        at org.apache.derby.jdbc.Driver30.newEmbedPreparedStatement(Unknown
Sour ce)
        at
org.apache.derby.impl.jdbc.EmbedConnection.prepareStatement(Unknown S
ource)
        at
org.apache.derby.impl.jdbc.EmbedConnection.prepareStatement(Unknown S
ource)
        at
coms.database.DatabaseConnection.executeSQL(DatabaseConnection.java:1 19)
        at
coms.database.DatabaseConnection.executeSQL(DatabaseConnection.java:9 8)
        at coms.database.Database.createTable(Database.java:200)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at
sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.
java:39) at
sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAcces
sorImpl.java:25)
        at java.lang.reflect.Method.invoke(Method.java:585)
        at
com.sun.jmx.mbeanserver.StandardMetaDataImpl.invoke(StandardMetaDataI
mpl.java:414)
        ... 32 more
```

Code 49: Create table bug stacktrace

After some error searching, it was revealed that the *getCreateTableSql(Table tbl)* in *Derby.java* had a bug. The primary key was set twice.

```
  /**
   * Adds a table to the database
   *
   * @param tbl
   *            The table to be added to the database
   * @return wheater the table was added successfully
   */
  protected String getCreateTableSql(Table tbl) throws SQLException,
ConnectionException {
    String def;
    TableField primary = null;
    primary = tbl.getPrimaryKey();
    String sqlStr = "CREATE TABLE " + tbl.getName() + "(";
     // Add primary key
     sqlStr += primary.getName() + " " + toSqlType(primary.getType());
//    sqlStr += " NOT NULL GENERATED ALWAYS AS IDENTITY (START WITH 1,
INCREMENT BY 1), PRIMARY KEY, ";
//    don't need to set primary key twice (see line 131)
     sqlStr += " NOT NULL GENERATED ALWAYS AS IDENTITY (START WITH 1,
INCREMENT BY 1), ";
     // Iterates over the fields in the table
     Iterator it = tbl.getFieldIterator();
    while (it.hasNext()) {
     TableField field = (TableField) it.next();
     // Adds all the non primary fields to the sql string
     if (!field.isPrimaryKey()) {
      sqlStr += field.getName() + " " + toSqlType(field.getType());
      // If there is a default value
      def = field.getDefault();
      if (def != null && !def.equals(""))
       sqlStr += " DEFAULT " + def;
      sqlStr += ", ";
     }
    }
    sqlStr += " PRIMARY KEY(" + primary.getName() + ")";
    sqlStr += ")";
    return sqlStr;
  }
```

Code 50: getCreateTable method from the Derby.java file on the APMS context manager

B.9 Bug fix: APMS MetaDataParser.java file

When deploying the JAR-file as a service on the context manager, the context manager unpacks the JAR-file and loads the components. But it also generates new meta files, based on the ones in the JAR-file. This generation, however, had some flaws. At some occurrences, seemingly random, the generation of new meta files were erroneous. After some error searching it was revealed that there was an error in the *MetaDataParser.java* file in the APMS context manager.

When parsing the meta files (SAX parsing), and the characters event is called, there's no guarantee that all the characters in the text element is available at the same time. For instance, when parsing the text element

<fieldtype>TIMESTAMP</fieldtype>

it's just as likely to get one *characters*-event with TIM and one with ESTAMP as to get just one event with TIMESTAMP. Therefore it makes sense to use the characters in a text element when the *endElement*-event is reached, since then all the characters have been read.

However, the *MetaDataParser.java* assumed that all the characters were available every time, and when it was not, erroneous meta files was created. To fix this, the functionality in the *characters()*-method was moved to the *endElement()*-method, and a *StringBuffer* was introduced in the *characters()*-method. This way it's guaranteed that all the characters has been received when performing operations.

```
public void characters(char[] text, int start, int length)
     throws SAXException {
  String value = new String(text, start, length);
  value = value.trim();

  if (starttag.equals("version"))
   data.setVersion(value);
  else if (starttag.equals("type"))
   data.setType(value);
  else if (starttag.equals("description"))
   data.setDescription(value);
  else if (starttag.equals("fieldname"))
   field.setName(value);
  else if (starttag.equals("fieldtype"))
    field.setType(value);
  else if (starttag.equals("default"))
    field.setDefault(value);
  else if (starttag.equals("dname"))
    data.addDependency(value);
  else if (starttag.equals("tablename"))
    table.setName(value);
  else if (starttag.equals("primarykey"))
    primaryKey = value;
  else if (starttag.equals("bindingtype"))
    data.setBindingType(value);
  else if (state.equals("config")) {
    if (value.length() > 0) {
      if (starttag.equals("optionname"))
        mapkey = value;
      if (starttag.equals("optionvalue"))
        mapval = value;
    }
  }
  else if (state.equals("subscribe")) {
    if (value.length() > 0)
      if (starttag.equals("to_table"))
        data.addSubscription(value);
  }
}
```

Code 51: MetaDataParser.java - The characters() method before the fix

```
StringBuffer tempValue = new StringBuffer();

public void characters(char[] text, int start, int length)
      throws SAXException {

    tempValue.append(new String(text, start, length));

  }
```

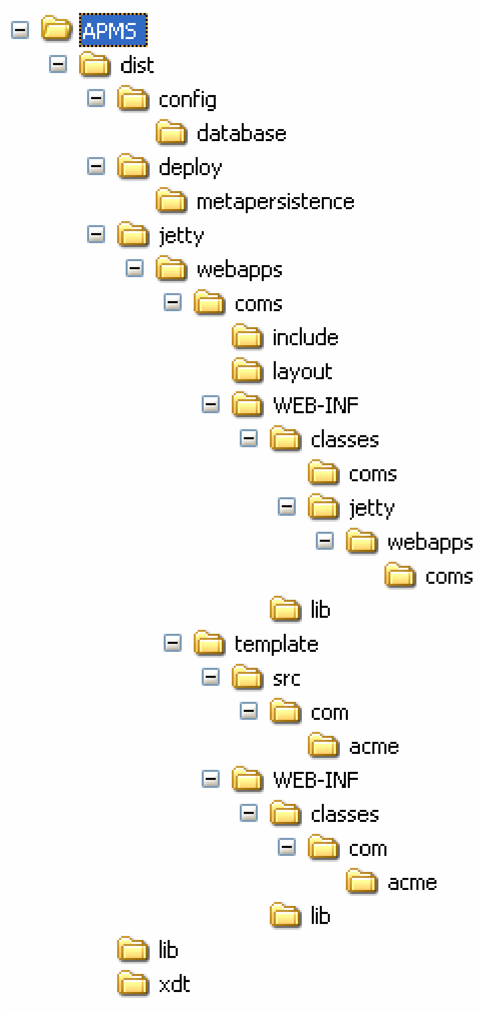
Code 52: MetaDataParser.java - The characters() method after the fix

Appendix C Installation and setup procedure

This appendix describes how to setup and launch the different system units.

C.1 Installing the APMS Context Manager

To install the APMS, simply extract the APMS directory from the supplied zip-file onto a hard drive. Verify that the directory structure is as follows:



```
APMS
    dist
        config
            database
        deploy
            metapersistence
        jetty
            webapps
                coms
                    include
                    layout
                    WEB-INF
                        classes
                            coms
                            jetty
                                webapps
                                    coms
                        lib
                template
                    src
                        com
                            acme
                    WEB-INF
                        classes
                            com
                                acme
                        lib
        lib
        xdt
```

Figure 58: APMS directory structure

Default port is currently set to 8080, but can be changed in the configuration file; APMS/dist/jetty/deploy_jetty.xml. The context manager can be started with this line from APMS/dist:

```
start java -Xmx1024m -Xms128m -cp lib\ant.jar;lib\antlr.jar;lib\commons-codec-1.3.jar;lib\commons-el.jar;lib\commons-logging.jar;lib\derby.jar;lib\derbyclient.jar;lib\derbytools.jar;lib\drools.jar;lib\fileupload.jar;lib\hsqldb.jar;lib\janino.jar;lib\jasper-compiler.jar;lib\jasper-runtime.jar;lib\javax.servlet.jar;lib\list.txt;lib\mx4j-remote.jar;lib\mx4j-tools.jar;lib\mx4j.jar;lib\OracleDriver.jar;lib\org.mortbay.jetty.jar;lib\org.mortbay.jmx.jar;lib\Serialio.jar;lib\tools.jar;lib\xercesImpl.jar;lib\xmlrpc.jar; CoMS.jar coms.CoMS
```

Alternativly, it can be started by launching the *run.bat* in the *bin/APMS* directory.

C.2 Adding the service to the APMS platform

If the precompiled version from the ZIP-file is used, the service will automatically be deployed in the APMS system. If not, this is how to deploy a service:

Open a browser and enter the server address (http://ip:port/RPC2) in the address bar. A web interface for the APMS context manager will now appear in the browser. Click on the *Browse…* button to find the jar-file containing the service. Select the jar-file and press *Open* and then *Submit* to upload it to the context manager. A message will appear saying; *All components were deployed successfully.*

To verify that the service is running, select *Manage Components* in the menu at the left. The new page will show a list of installed components and their state (*Running* or *Stopped*).

To remove the service, select *Manage Packages* from the menu at the left. A list of deployed packages will appear on the new page. To remove a service click on *Delete* next to the package name.

C.3 Launching the FiFaMoS context source

To add sensor values to the APMS context management system, a context source must be present. This can either be the FiFaMoS Context Consumer IMlet running on the M2M module or the Aplicom 12 IMP concept simulator, or the sensor simulator. If the M2M module shall be used, various parameters need to be set correctly. These parameters can be altered with the Aplicom N12i configurator, and is as follows:

```
m2msystem:N1=255,
N3=10,
T3=10,
BaudRate=4,
supplementaryservice:AutoAnswer=0,
CallWaiting=0,
CallingLineIdentificationRestriction=0,

sms:RemoveOldestMessage=1,
ServiceCentreAddress=+4790002100,
MessageDeliveryReports=0,
MessageReplyPath=0,
MessageValidityPeriod=5,

wapcommon:CSDNNumberAuthentication=0,
IncomingCHAPAuthentication=0,
IncomingCHAPUsername=dj,
IncomingCHAPPassword=dj,
DefaultConnectionIndex=0,
GprsAlwaysOnline=1,
HLAPort=0,
HLAName=,
UseM2MGateway=0,

wapconnection0:WTPPortNumber=0,
WAPBearer=1,
DestinationAddress=,
GatewayIPAddress=0.0.0.0,
WAPGWPortNumber=1,
CHAPUserName=,
CHAPPassword=,
GatewayAddress=internet,
DataCallBitrate=0,
ConnectionTimeout=0,

wapconnection1:WTPPortNumber=65535,
WAPBearer=5,
GatewayAddress=,
DestinationAddress=,
GatewayIPAddress=0.0.0.0,
WAPGWPortNumber=65535,
CHAPUserName=,
CHAPPassword=,
DataCallBitrate=0,
ConnectionTimeout=0,
wapconnection2:WTPPortNumber=65535,
WAPBearer=5,
GatewayAddress=,
DestinationAddress=,
GatewayIPAddress=0.0.0.0,
WAPGWPortNumber=65535,
CHAPUserName=,
CHAPPassword=,
DataCallBitrate=0,
ConnectionTimeout=0,

wapconnection3:WTPPortNumber=65535,
WAPBearer=5,
GatewayAddress=,
DestinationAddress=,
GatewayIPAddress=0.0.0.0,
WAPGWPortNumber=65535,
CHAPUserName=,
CHAPPassword=,
DataCallBitrate=0,
ConnectionTimeout=0,

wapconnection4:WTPPortNumber=65535,
WAPBearer=5,
GatewayAddress=,
DestinationAddress=,
GatewayIPAddress=0.0.0.0,
WAPGWPortNumber=65535,
CHAPUserName=,
CHAPPassword=,
DataCallBitrate=0,
ConnectionTimeout=0,

serverioinput:AlarmMode=1,
serveriooutput:Output1=1,
Output2=0,
Output3=0,
Output4=0,
Output5=0,
Output6=0,
Output7=0,
Output8=0,
Output9=0,

serverioport:Port1=4,
Port3=1,
BaudRate=8,
GLL=1,
GGA=1,
RMC=1,
VTG=1,

serveriohttp:Data=1,
LoginType=0,
Username=,
Password=,
DialupNumber=,
ProxyName=,
DataCallType=-1,
DataCallSpeed=0,
SessionSecurity=1,
AccessPointName=internet,

audio:AudioMode=0,
EchoMode=0,

userio:Identifier=~MasterNumber=~Password=[[b5f1]]~Aliases=~DisableAck=0~
operatingmode:
imletdownloading:IPAddress=0.0.0.0,
```

Check that all switches are as indicated Figure 59 and that the N12i is properly connected. Connect the GPS to the test board at port 3 using the crossed gender changer, connect the GSM antenna to the N12i and finally connect a serial cable between a computer and the N12i test board port 2. Port 1 may be used for logging, if so, connect a serial cable between a computer and port 1 on the test board. To get debug information, a program for reading from the computers serial port has to be used. This is called *LoggingServer* and is located in the attached ZIP-file.

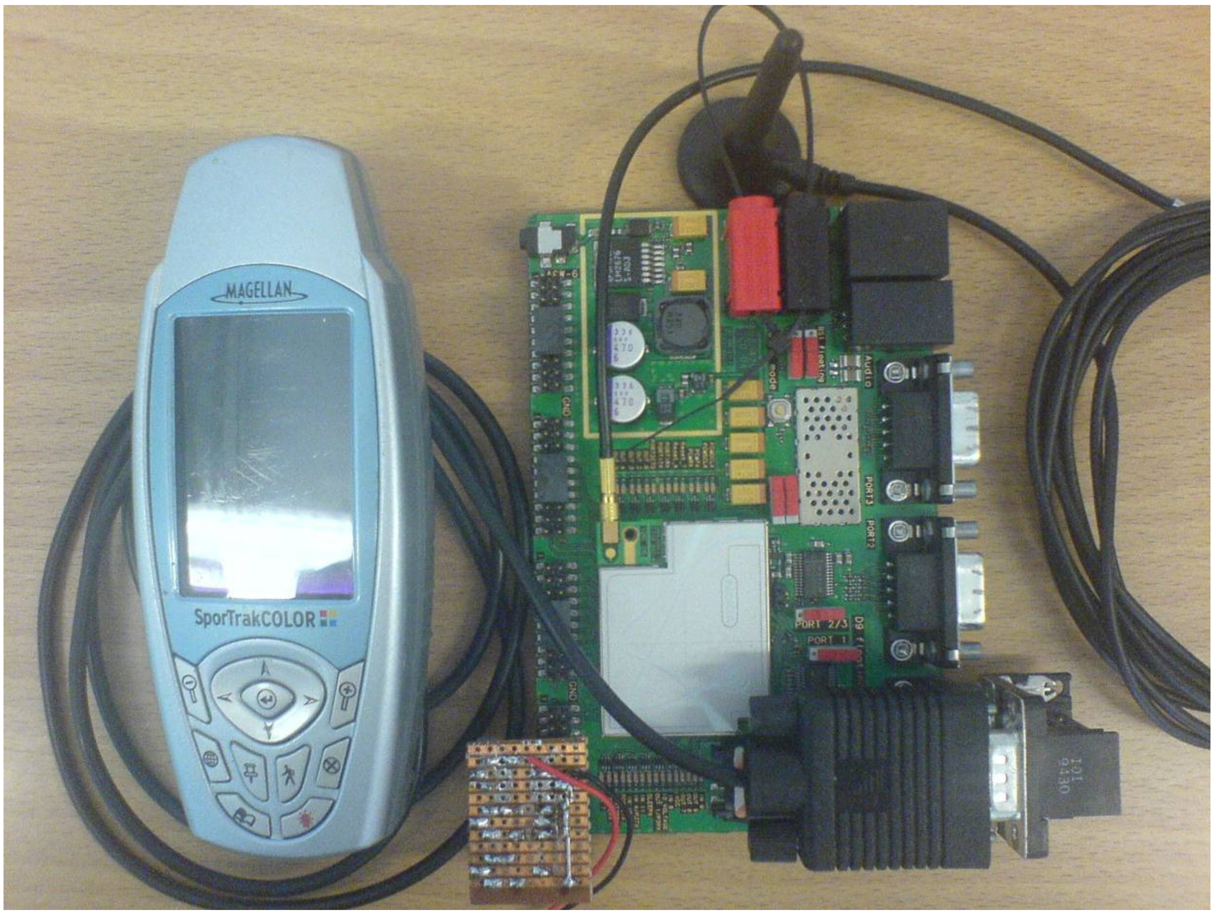


Figure 59: The Aplicom N12i module with test board and GPS

Install and start the Aplicom Configurator and power on the test board. Select *IMlet Loading -> Cable* from the menu. Browse and select the jar-file, and press [->] to upload it. Select the program from the list at far right and press *Start* to start the program.

If the sensor simulator shall be used instead of the M2M IMlet, launch the *run.bat* file in the FiFaMoS Context Source Simulator folder.

C.4 Installing and configuring the FiFaMoS Context Consumer

To launch the FiFaMoS Context Consumer, launch the *run.bat* file in the */bin/FiFaMoS Consumer Consumer/* folder of the ZIP-file can be run. If an error message occurs in the status text box, check that the correct server address is set in the settings dialog. If the program works, but no fish frames or sea cages occur in the navigation tree, they have to be added manually in the settings dialog.

C.5 Installing the FiFaMoS Mobile Context Consumer

Transfer the *FiFaMoS_Mobile.jar* from the */bin/ FiFaMoS Mobile Context Consumer* folder to a mobile phone via IR or Bluetooth, and install the MIDlet on the phone. When the MIDlet is launched, press start. If the correct server address is given in the settings, a list of the frames will be displayed, and it will be possible to navigate to the sea cages and sensors. The .jar file can also be launced in the Wireless Toolkit mobile simulator.

C.6 Testing FiFaMoS

To get the system working, the fish farm must be created in the FiFaMoS Context Consumer. This is done in a settings dialog (see *the settings dialog* section under Appendix A). Here it is possible to create frames, sea cages and sensors. It is important that the IMEI-number of the M2M module is set correctly when creating the sea cage. This is the number that is used to identify the sea cage and can be found underneath the physical M2M module. For demonstrative purposes, the button bottom left can be used to create a number of frames, sea cages and sensors. The IMEI number of the M2M module used when testing will automatically be assigned to the first sea cage.

When the M2M application is loaded into the module, and the sensors are correctly connected, the module will automatically connect to the context manager (assuming that the correct server-address is set in the application, to change this, one would have to recompile the source code). The context manager will check the IMEI number of the sea cage, and generate an XML-file that describes the semantics of the sea cage, including information of the sensors and other parameters. This information will be used to configure the M2M module. When the module is configured, it will automatically start to send its sensor values to the context manager in given intervals. The context manager stores this information in the database, and makes it available to the clients. The values are also checked to see if they are within given bounds. If not, alarm information is stored in the response XML sent back to the module. This alarm information is used to generate an SMS message that is sent from the module to the operator of the frame. Alarms are also stored in the database, so that they can be displayed in the clients.

To view the sensor data, either the FiFaMoS Context Consumer or the FiFaMoS Mobile Context Consumer can be used. These are described in Appendix A, and operation is self-explanatory.

Appendix D Hardware

D.1 The dummy sensors

Since we were not able to get real sensors from the aquaculture industry, we decided to use some adjustable resistors instead. By connecting these to a battery, and attaching them to the Aplicom N12i test board, sensor values could be adjusted by trimming the resistors.

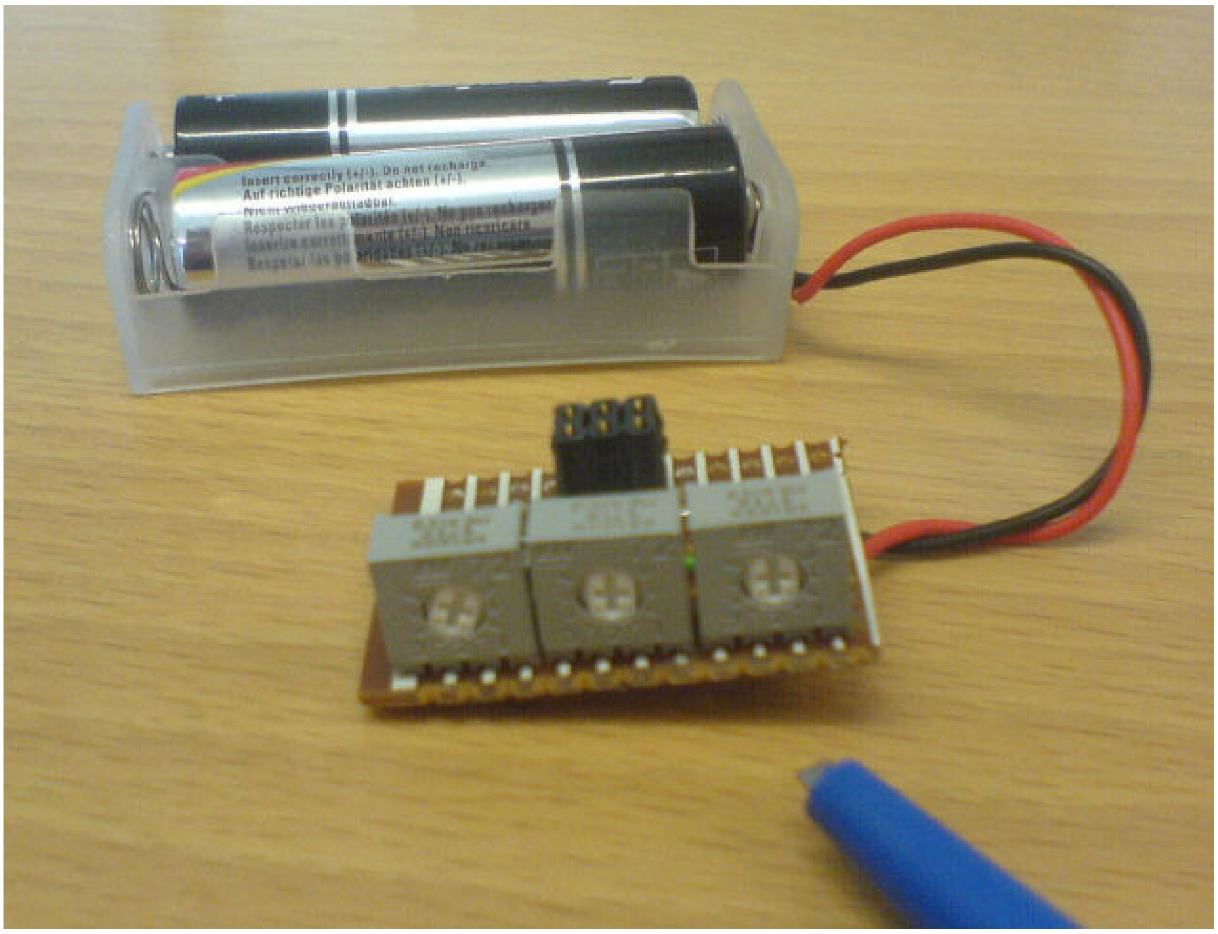


Figure 60: Dummy sensors for the Aplicom N12i M2M module

D.2 Camera connector

Since the M2M module only had serial ports for connecting a camera, we needed a camera using this kind of interface. There are not many of these, but we found one in a fleet management system called TraceMe. This camera had a 4-pin port that seldom is used for serial purposes, so we had to make an adapter. Using a sound cable from an old computer, and a 9-pin male serial-port, the adapter was realized and put into action.

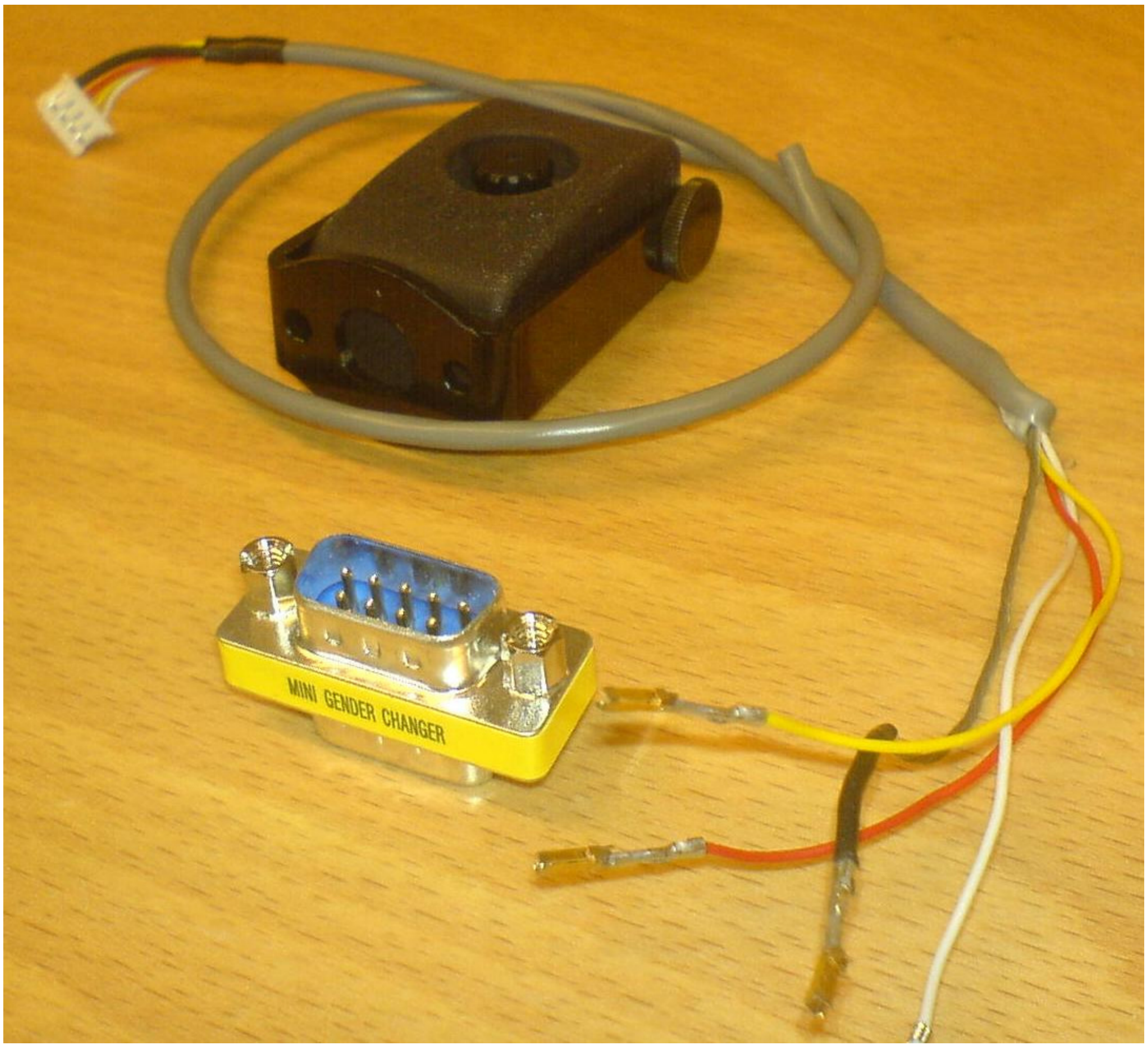


Figure 61: The camera and camera connector

D.3 GPS connector

Since the serial port on the Aplicom N12i has the opposite gender as the ones on PCs, we could not connect the GPS. Therefore we had to make an adapter. This adapter also had to cross the RX and TX pins, so that data sent from the GPS could be read on the Aplicom N12i.

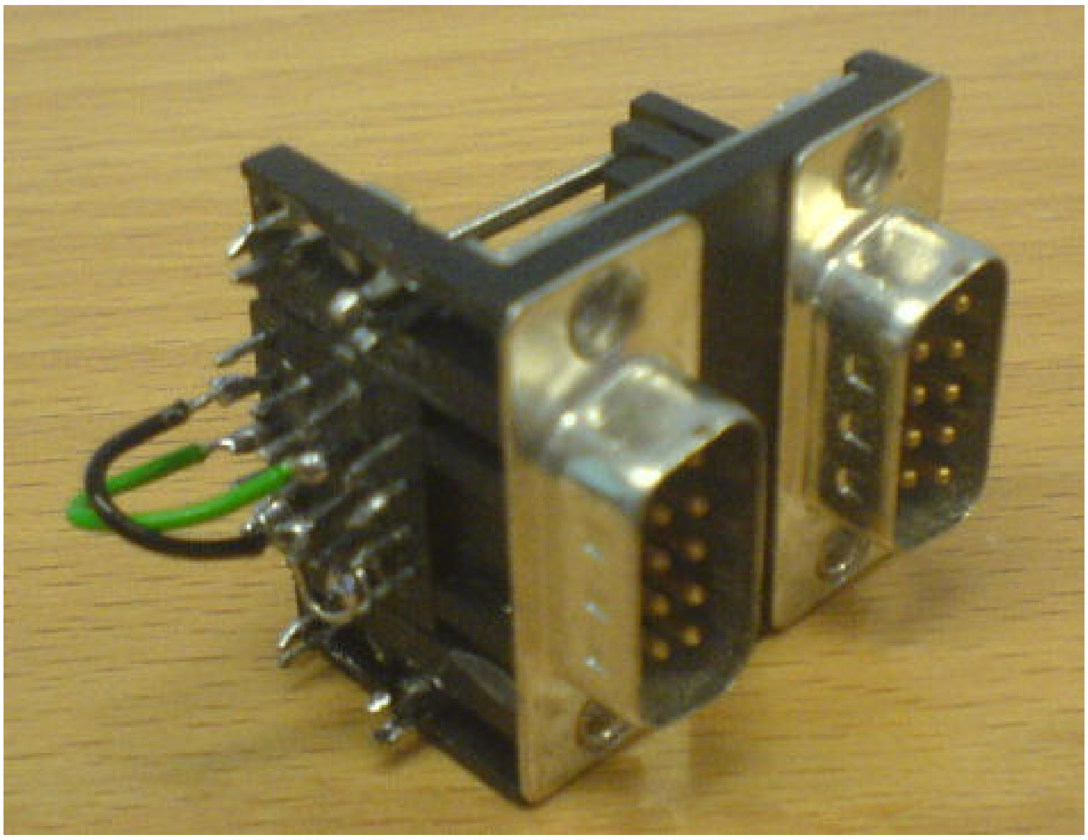


Figure 62: GPS to Aplicom gender changing adapter

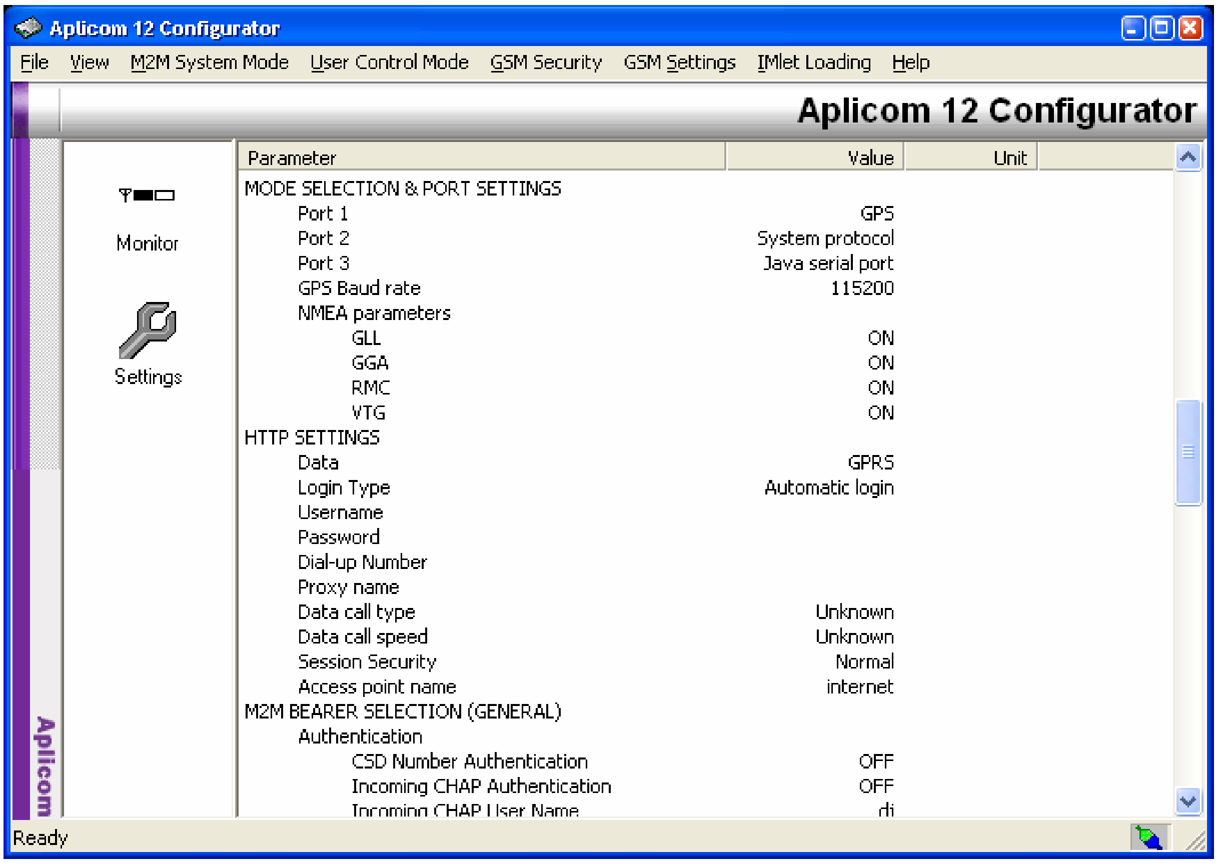
The adapter was made from some serial ports from an old computer. The printed circuit board were removed, and replaced with short cables connecting the two serial ports. When attaching this adapter to the Aplicom N12i serial port, the gender was changed from female to male, and the GPS could be connected.

Appendix E Software

This appendix show the software related to the Aplicom N12i M2M GPRS module. In addition the JMX monitor for testing and debugging services on the APMS platform is presented.

E.1 *Aplicom N12i configurator*

The Aplicom N12i configurator is used for setting various parameters on the module, and uploading and starting applications. Parameters that can be set and read includes among others: GPRS settings and serial port settings.

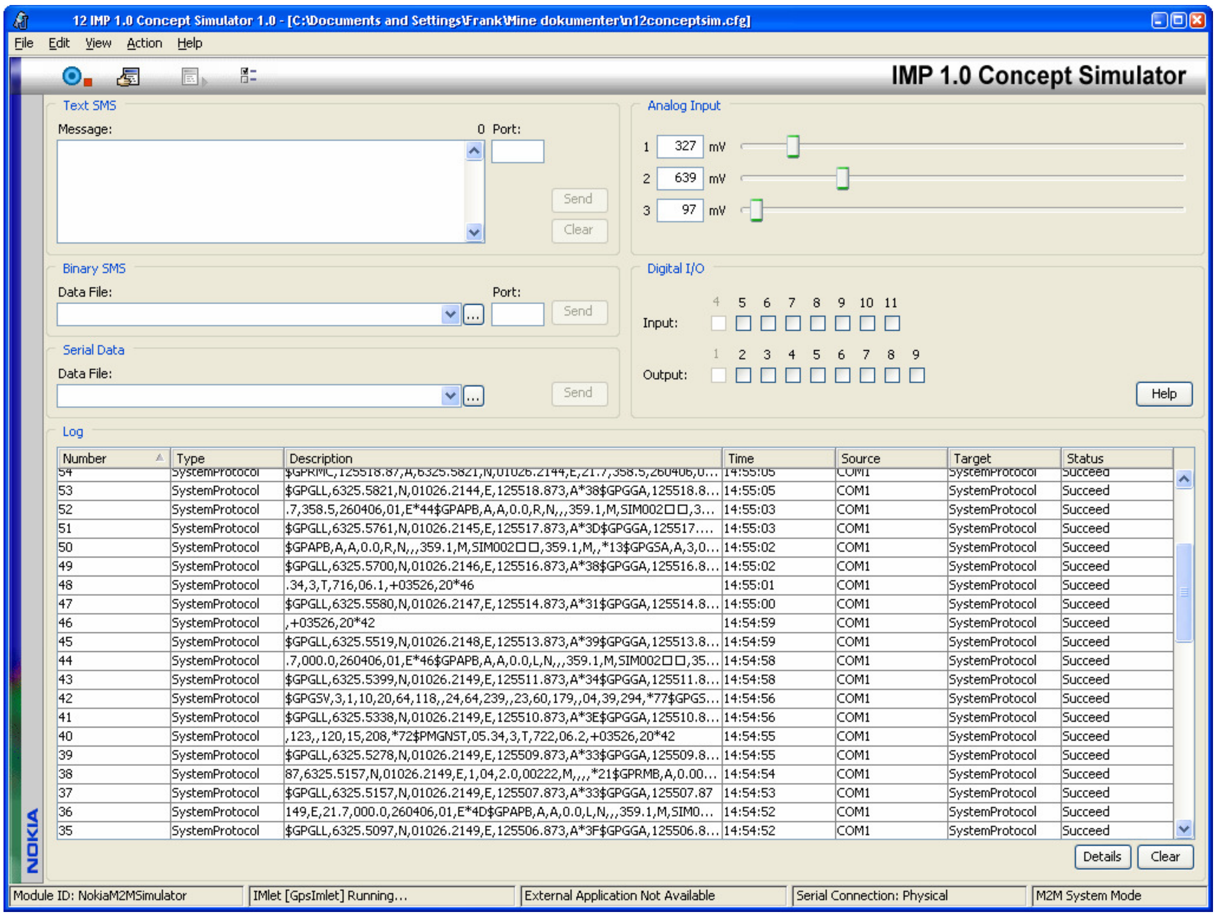

Aplicom 12 Configurator
File View M2M System Mode User Control Mode GSM Security GSM Settings IMlet Loading Help
Aplicom 12 Configurator
Monitor
Settings
Aplicom
Ready

Parameter	Value	Unit
MODE SELECTION & PORT SETTINGS		
Port 1	GPS	
Port 2	System protocol	
Port 3	Java serial port	
GPS Baud rate	115200	
NMEA parameters		
GLL	ON	
GGA	ON	
RMC	ON	
VTG	ON	
HTTP SETTINGS		
Data	GPRS	
Login Type	Automatic login	
Username		
Password		
Dial-up Number		
Proxy name		
Data call type	Unknown	
Data call speed	Unknown	
Session Security	Normal	
Access point name	internet	
M2M BEARER SELECTION (GENERAL)		
Authentication		
CSD Number Authentication	OFF	
Incoming CHAP Authentication	OFF	
Incoming CHAP User Name	di	

Figure 63: The Aplicom N12i Configurator

E.2 Aplicom N12i IMP 1.0 Concept Simulator

The Aplicom N12i IMP 1.0 Concept Simulator is an application that emulates the Aplicom N12i functionality. MIDlets programmed for the module can be tested here directly from Eclipse, and the various inputs and outputs are available in the GUI in addition to logging information. It is also possible to simulate the SMS functionality.



1	327	mV
2	639	mV
3	97	mV

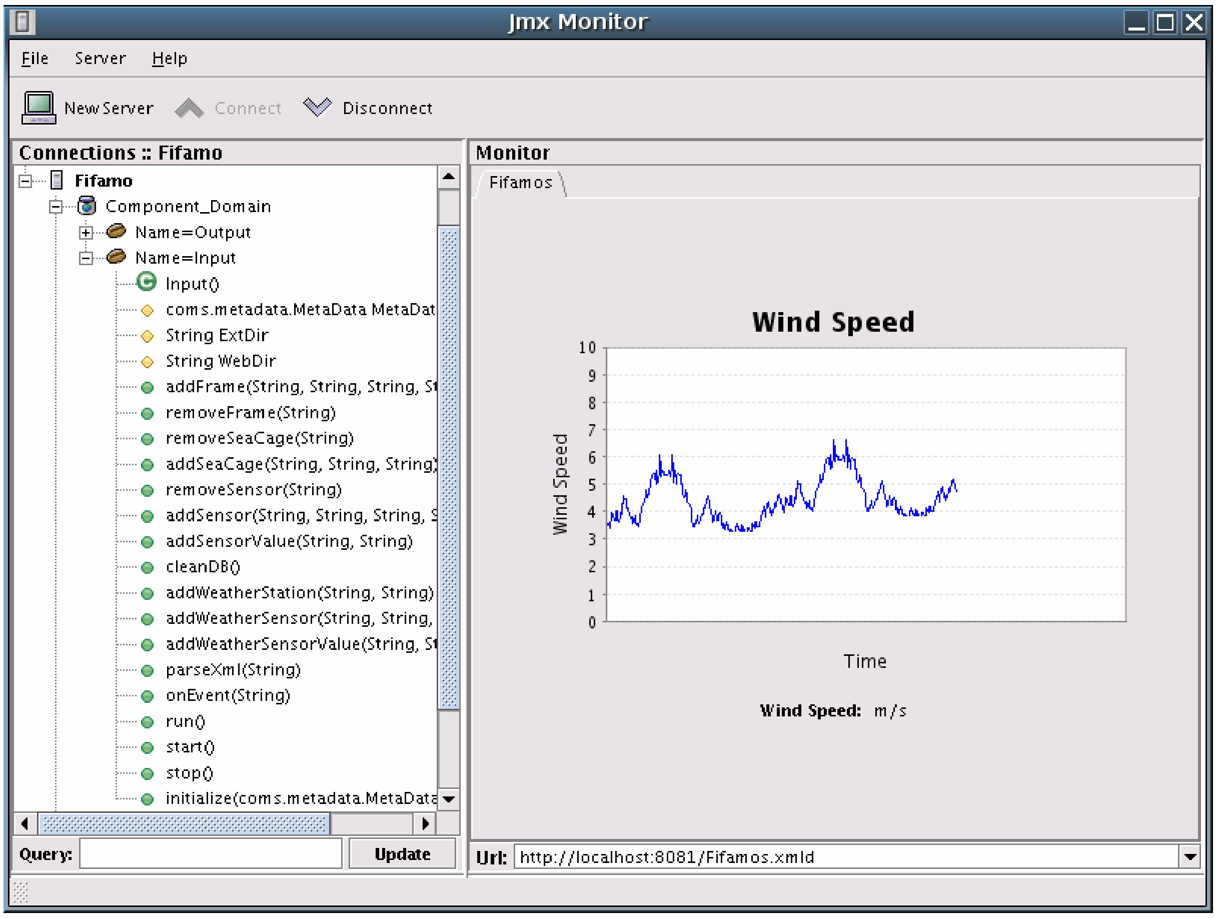
Number	Type	Description	Time	Source	Target	Status
54	SystemProtocol	$GPRMC,125518.87,A,6325.5821,N,01026.2144,E,21.7,358.5,260406,0…	14:55:05	COM1	SystemProtocol	Succeed
53	SystemProtocol	$GPGLL,6325.5821,N,01026.2144,E,125518.873,A*38$GPGGA,125518.8…	14:55:05	COM1	SystemProtocol	Succeed
52	SystemProtocol	.7,358.5,260406,01,E*44$GPAPB,A,A,0.0,R,N,,,359.1,M,SIM002□□,3…	14:55:03	COM1	SystemProtocol	Succeed
51	SystemProtocol	$GPGLL,6325.5761,N,01026.2145,E,125517.873,A*3D$GPGGA,125517.…	14:55:03	COM1	SystemProtocol	Succeed
50	SystemProtocol	$GPAPB,A,A,0.0,R,N,,,359.1,M,SIM002□□,359.1,M,,*13$GPGSA,A,3,0…	14:55:02	COM1	SystemProtocol	Succeed
49	SystemProtocol	$GPGLL,6325.5700,N,01026.2146,E,125516.873,A*38$GPGGA,125516.8…	14:55:02	COM1	SystemProtocol	Succeed
48	SystemProtocol	.34,3,T,716,06.1,+03526,20*46	14:55:01	COM1	SystemProtocol	Succeed
47	SystemProtocol	$GPGLL,6325.5580,N,01026.2147,E,125514.873,A*31$GPGGA,125514.8…	14:55:00	COM1	SystemProtocol	Succeed
46	SystemProtocol	,+03526,20*42	14:54:59	COM1	SystemProtocol	Succeed
45	SystemProtocol	$GPGLL,6325.5519,N,01026.2148,E,125513.873,A*39$GPGGA,125513.8…	14:54:59	COM1	SystemProtocol	Succeed
44	SystemProtocol	.7,000.0,260406,01,E*46$GPAPB,A,A,0.0,L,N,,,359.1,M,SIM002□□,35…	14:54:58	COM1	SystemProtocol	Succeed
43	SystemProtocol	$GPGLL,6325.5399,N,01026.2149,E,125511.873,A*34$GPGGA,125511.8…	14:54:58	COM1	SystemProtocol	Succeed
42	SystemProtocol	$GPGSV,3,1,10,20,64,118,,24,64,239,,23,60,179,,04,39,294,*77$GPGS…	14:54:56	COM1	SystemProtocol	Succeed
41	SystemProtocol	$GPGLL,6325.5338,N,01026.2149,E,125510.873,A*3E$GPGGA,125510.8…	14:54:56	COM1	SystemProtocol	Succeed
40	SystemProtocol	,123,,120,15,208,*72$PMGNST,05.34,3,T,722,06.2,+03526,20*42	14:54:55	COM1	SystemProtocol	Succeed
39	SystemProtocol	$GPGLL,6325.5278,N,01026.2149,E,125509.873,A*33$GPGGA,125509.8…	14:54:55	COM1	SystemProtocol	Succeed
38	SystemProtocol	87,6325.5157,N,01026.2149,E,1,04,2.0,00222,M,,,,*21$GPRMB,A,0.00…	14:54:54	COM1	SystemProtocol	Succeed
37	SystemProtocol	$GPGLL,6325.5157,N,01026.2149,E,125507.873,A*33$GPGGA,125507.87	14:54:53	COM1	SystemProtocol	Succeed
36	SystemProtocol	149,E,21.7,000.0,260406,01,E*4D$GPAPB,A,A,0.0,L,N,,,359.1,M,SIM0…	14:54:52	COM1	SystemProtocol	Succeed
35	SystemProtocol	$GPGLL,6325.5097,N,01026.2149,E,125506.873,A*3F$GPGGA,125506.8…	14:54:52	COM1	SystemProtocol	Succeed

Module ID: NokiaM2MSimulator | IMlet [GpsImlet] Running… | External Application Not Available | Serial Connection: Physical | M2M System Mode

Figure 64: The Aplicom N12i IMP 1.0 Concept Simulator

E.3 The JMX monitor

In this project, a JMX monitor is used to test the service on the APMS context manager. This particular monitor is created by Dan Peder Eriksen, at The University of Tromsø.



```
Fifamo
  Component_Domain
    Name=Output
    Name=Input
      Input()
      coms.metadata.MetaData MetaDat
      String ExtDir
      String WebDir
      addFrame(String, String, String, St
      removeFrame(String)
      removeSeaCage(String)
      addSeaCage(String, String, String)
      removeSensor(String)
      addSensor(String, String, String, S
      addSensorValue(String, String)
      cleanDB()
      addWeatherStation(String, String)
      addWeatherSensor(String, String,
      addWeatherSensorValue(String, St
      parseXml(String)
      onEvent(String)
      run()
      start()
      stop()
      initialize(coms.metadata.MetaData
```

Url: http://localhost:8081/Fifamos.xmld

Figure 65: The JMX monitor

The monitor is quite easy in use; Connect to a managed server (in this case *FiFaMoS*), select a managed bean (here; *Input*) and take use of the available methods and variables. Methods can be invoked, but only String is supported as argument(s) in this version.

Appendix F Class diagrams

F.1 Classes common for all applications

com.telenor.apms.fifamos.objects

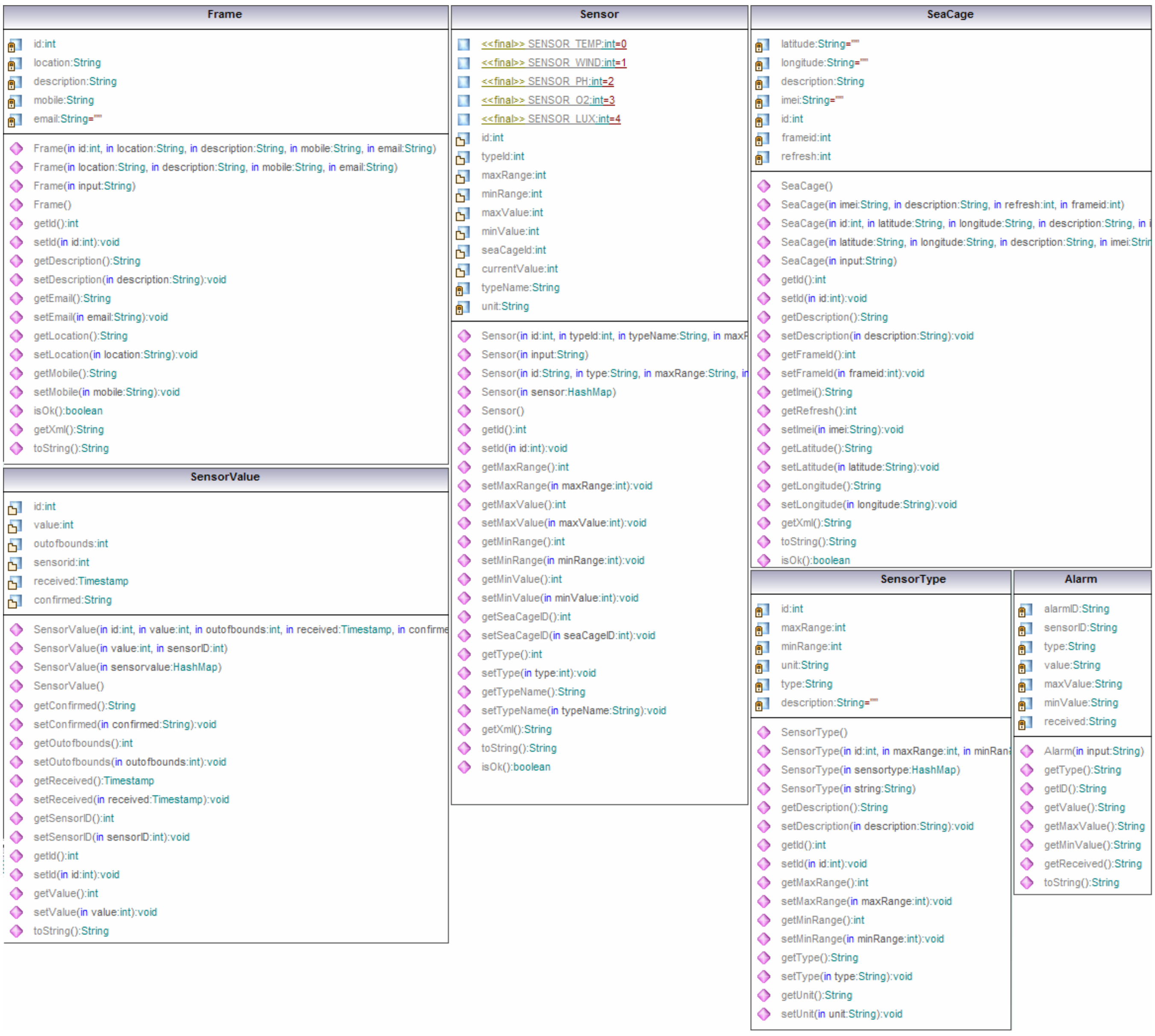

Frame
id:int
location:String
description:String
mobile:String
email:String=""
Frame(in id:int, in location:String, in description:String, in mobile:String, in email:String)
Frame(in location:String, in description:String, in mobile:String, in email:String)
Frame(in input:String)
Frame()
getId():int
setId(in id:int):void
getDescription():String
setDescription(in description:String):void
getEmail():String
setEmail(in email:String):void
getLocation():String
setLocation(in location:String):void
getMobile():String
setMobile(in mobile:String):void
isOk():boolean
getXml():String
toString():String
SensorValue
id:int
value:int
outofbounds:int
sensorid:int
received:Timestamp
confirmed:String
SensorValue(in id:int, in value:int, in outofbounds:int, in received:Timestamp, in confirme
SensorValue(in value:int, in sensorID:int)
SensorValue(in sensorvalue:HashMap)
SensorValue()
getConfirmed():String
setConfirmed(in confirmed:String):void
getOutofbounds():int
setOutofbounds(in outofbounds:int):void
getReceived():Timestamp
setReceived(in received:Timestamp):void
getSensorID():int
setSensorID(in sensorID:int):void
getId():int
setId(in id:int):void
getValue():int
setValue(in value:int):void
toString():String
Sensor
<<final>> SENSOR_TEMP:int=0
<<final>> SENSOR_WIND:int=1
<<final>> SENSOR_PH:int=2
<<final>> SENSOR_O2:int=3
<<final>> SENSOR_LUX:int=4
id:int
typeId:int
maxRange:int
minRange:int
maxValue:int
minValue:int
seaCageId:int
currentValue:int
typeName:String
unit:String
Sensor(in id:int, in typeId:int, in typeName:String, in maxR
Sensor(in input:String)
Sensor(in id:String, in type:String, in maxRange:String, in
Sensor(in sensor:HashMap)
Sensor()
getId():int
setId(in id:int):void
getMaxRange():int
setMaxRange(in maxRange:int):void
getMaxValue():int
setMaxValue(in maxValue:int):void
getMinRange():int
setMinRange(in minRange:int):void
getMinValue():int
setMinValue(in minValue:int):void
getSeaCageID():int
setSeaCageID(in seaCageID:int):void
getType():int
setType(in type:int):void
getTypeName():String
setTypeName(in typeName:String):void
getXml():String
toString():String
isOk():boolean
SeaCage
latitude:String=""
longitude:String=""
description:String
imei:String=""
id:int
frameid:int
refresh:int
SeaCage()
SeaCage(in imei:String, in description:String, in refresh:int, in frameid:int)
SeaCage(in id:int, in latitude:String, in longitude:String, in description:String, in i
SeaCage(in latitude:String, in longitude:String, in description:String, in imei:Strin
SeaCage(in input:String)
getId():int
setId(in id:int):void
getDescription():String
setDescription(in description:String):void
getFrameId():int
setFrameId(in frameid:int):void
getImei():String
getRefresh():int
setImei(in imei:String):void
getLatitude():String
setLatitude(in latitude:String):void
getLongitude():String
setLongitude(in longitude:String):void
getXml():String
toString():String
isOk():boolean
SensorType
id:int
maxRange:int
minRange:int
unit:String
type:String
description:String=""
SensorType()
SensorType(in id:int, in maxRange:int, in minRan
SensorType(in sensortype:HashMap)
SensorType(in string:String)
getDescription():String
setDescription(in description:String):void
getId():int
setId(in id:int):void
getMaxRange():int
setMaxRange(in maxRange:int):void
getMinRange():int
setMinRange(in minRange:int):void
getType():String
setType(in type:String):void
getUnit():String
setUnit(in unit:String):void
Alarm
alarmID:String
sensorID:String
type:String
value:String
maxValue:String
minValue:String
received:String
Alarm(in input:String)
getType():String
getID():String
getValue():String
getMaxValue():String
getMinValue():String
getReceived():String
toString():String

Figure 66: com.telenor.apms.fifamos.objects class diagram

F.2 FiFaMoS APMS Context Manager service

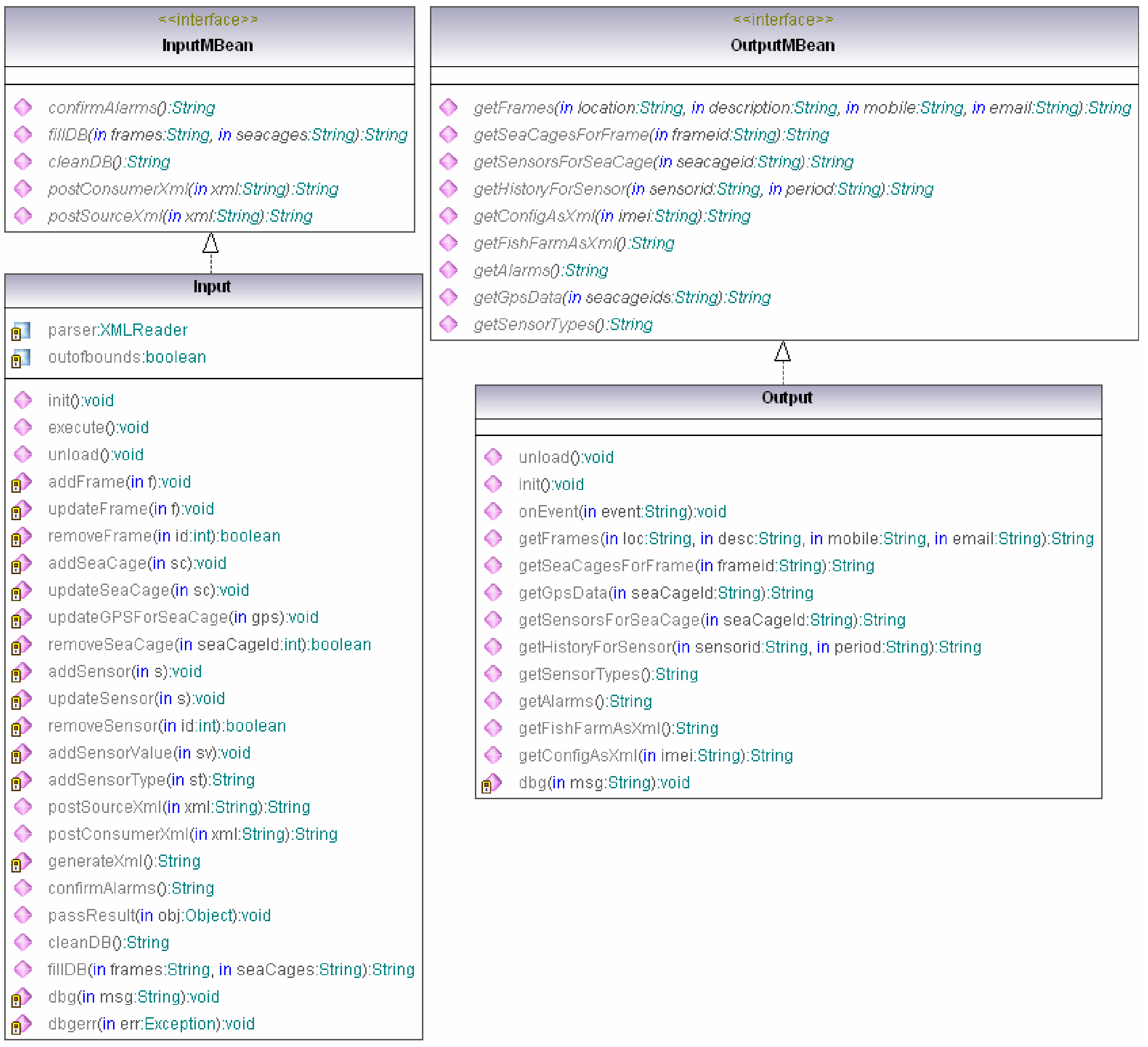

<<interface>>
InputMBean
confirmAlarms():String
fillDB(in frames:String, in seacages:String):String
cleanDB():String
postConsumerXml(in xml:String):String
postSourceXml(in xml:String):String
<<interface>>
OutputMBean
getFrames(in location:String, in description:String, in mobile:String, in email:String):String
getSeaCagesForFrame(in frameid:String):String
getSensorsForSeaCage(in seacageid:String):String
getHistoryForSensor(in sensorid:String, in period:String):String
getConfigAsXml(in imei:String):String
getFishFarmAsXml():String
getAlarms():String
getGpsData(in seacageids:String):String
getSensorTypes():String
Input
parser:XMLReader
outofbounds:boolean
init():void
execute():void
unload():void
addFrame(in f):void
updateFrame(in f):void
removeFrame(in id:int):boolean
addSeaCage(in sc):void
updateSeaCage(in sc):void
updateGPSForSeaCage(in gps):void
removeSeaCage(in seaCageId:int):boolean
addSensor(in s):void
updateSensor(in s):void
removeSensor(in id:int):boolean
addSensorValue(in sv):void
addSensorType(in st):String
postSourceXml(in xml:String):String
postConsumerXml(in xml:String):String
generateXml():String
confirmAlarms():String
passResult(in obj:Object):void
cleanDB():String
fillDB(in frames:String, in seaCages:String):String
dbg(in msg:String):void
dbgerr(in err:Exception):void
Output
unload():void
init():void
onEvent(in event:String):void
getFrames(in loc:String, in desc:String, in mobile:String, in email:String):String
getSeaCagesForFrame(in frameid:String):String
getGpsData(in seaCageId:String):String
getSensorsForSeaCage(in seaCageId:String):String
getHistoryForSensor(in sensorid:String, in period:String):String
getSensorTypes():String
getAlarms():String
getFishFarmAsXml():String
getConfigAsXml(in imei:String):String
dbg(in msg:String):void

Figure 67: com.telenor.apms.fifamos class diagram

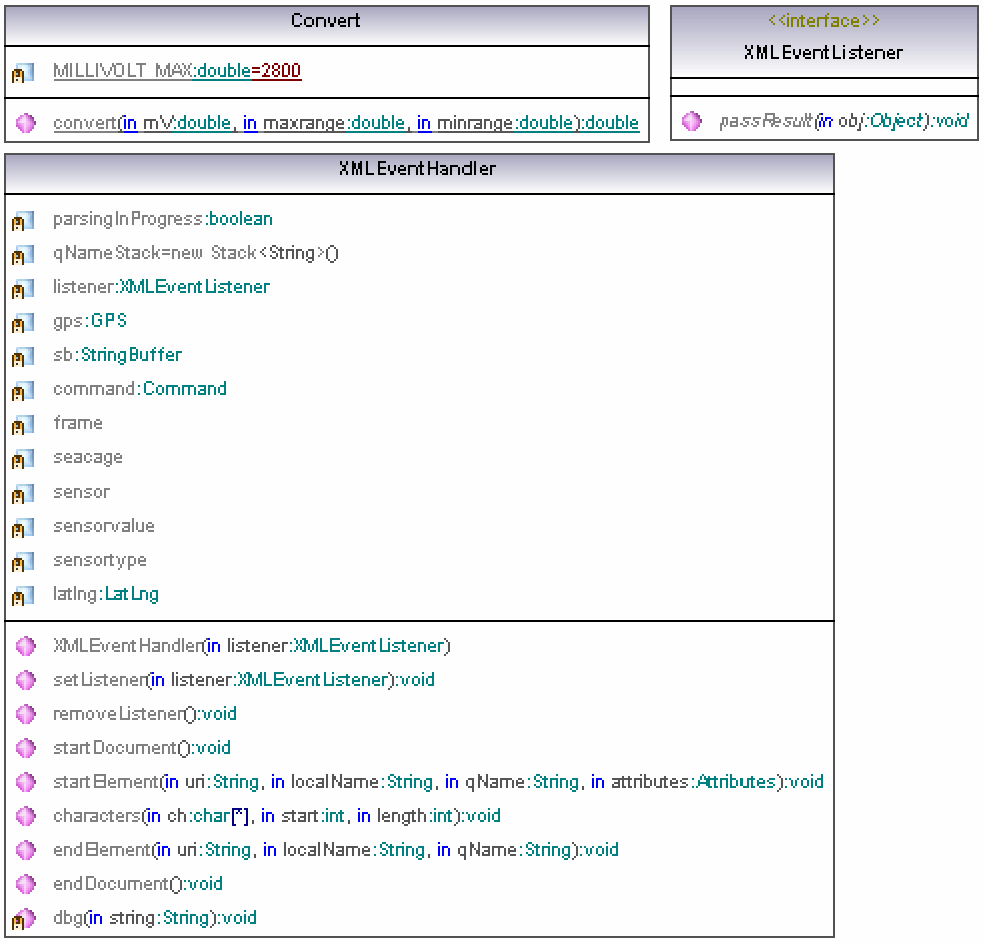
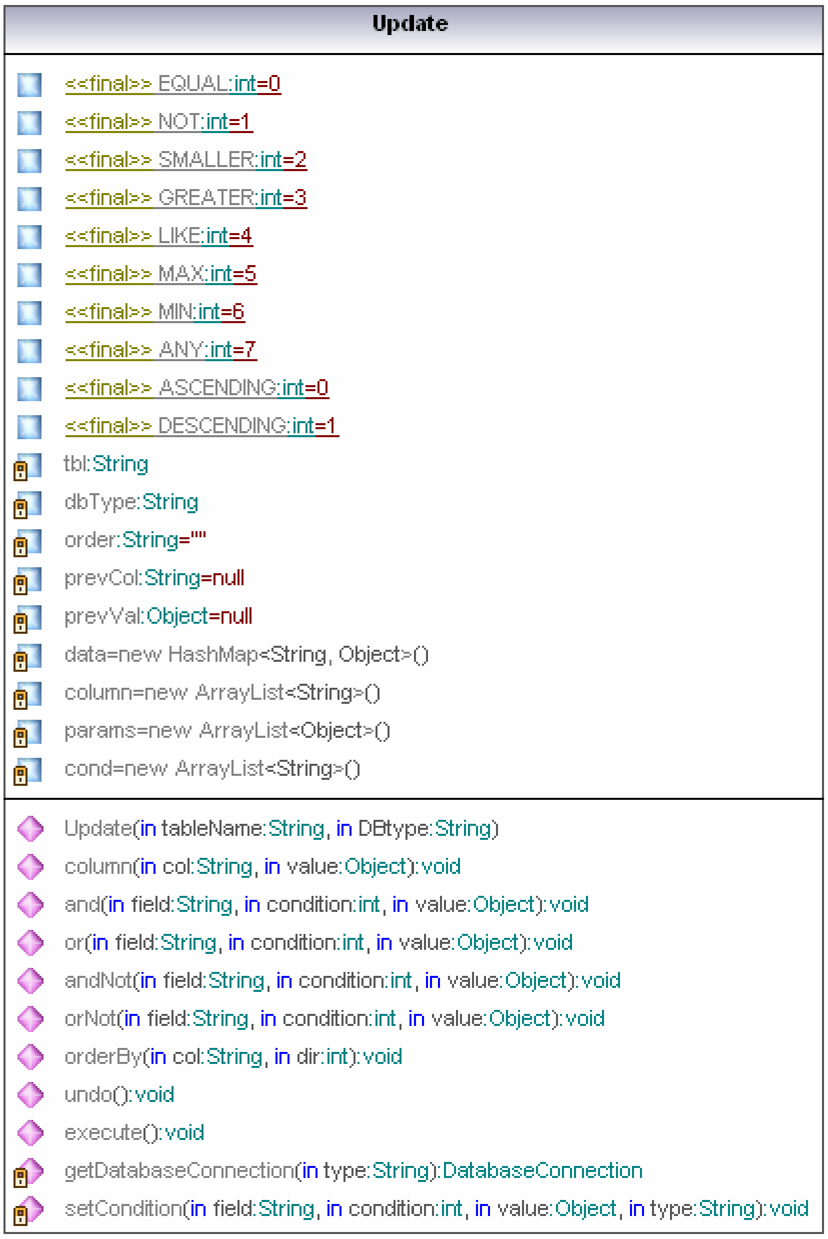

Convert
MILLIVOLT_MAX:double=2800
convert(in mV:double, in maxrange:double, in minrange:double):double
<<interface>>
XMLEventListener
passResult(in obj:Object):void
XMLEventHandler
parsingInProgress:boolean
qNameStack=new Stack<String>()
listener:XMLEventListener
gps:GPS
sb:StringBuffer
command:Command
frame
seacage
sensor
sensorvalue
sensortype
latlng:LatLng
XMLEventHandler(in listener:XMLEventListener)
setListener(in listener:XMLEventListener):void
removeListener():void
startDocument():void
startElement(in uri:String, in localName:String, in qName:String, in attributes:Attributes):void
characters(in ch:char[*], in start:int, in length:int):void
endElement(in uri:String, in localName:String, in qName:String):void
endDocument():void
dbg(in string:String):void

Figure 68: com.telenor.apms.fifamos.utils class diagram



```
Update

<<final>> EQUAL:int=0
<<final>> NOT:int=1
<<final>> SMALLER:int=2
<<final>> GREATER:int=3
<<final>> LIKE:int=4
<<final>> MAX:int=5
<<final>> MIN:int=6
<<final>> ANY:int=7
<<final>> ASCENDING:int=0
<<final>> DESCENDING:int=1
tbl:String
dbType:String
order:String=""
prevCol:String=null
prevVal:Object=null
data=new HashMap<String, Object>()
column=new ArrayList<String>()
params=new ArrayList<Object>()
cond=new ArrayList<String>()

Update(in tableName:String, in DBtype:String)
column(in col:String, in value:Object):void
and(in field:String, in condition:int, in value:Object):void
or(in field:String, in condition:int, in value:Object):void
andNot(in field:String, in condition:int, in value:Object):void
orNot(in field:String, in condition:int, in value:Object):void
orderBy(in col:String, in dir:int):void
undo():void
execute():void
getDatabaseConnection(in type:String):DatabaseConnection
setCondition(in field:String, in condition:int, in value:Object, in type:String):void
```

Figure 69: coms.tools.relational.Update class diagram

F.3 The FiFaMoS Context Source application

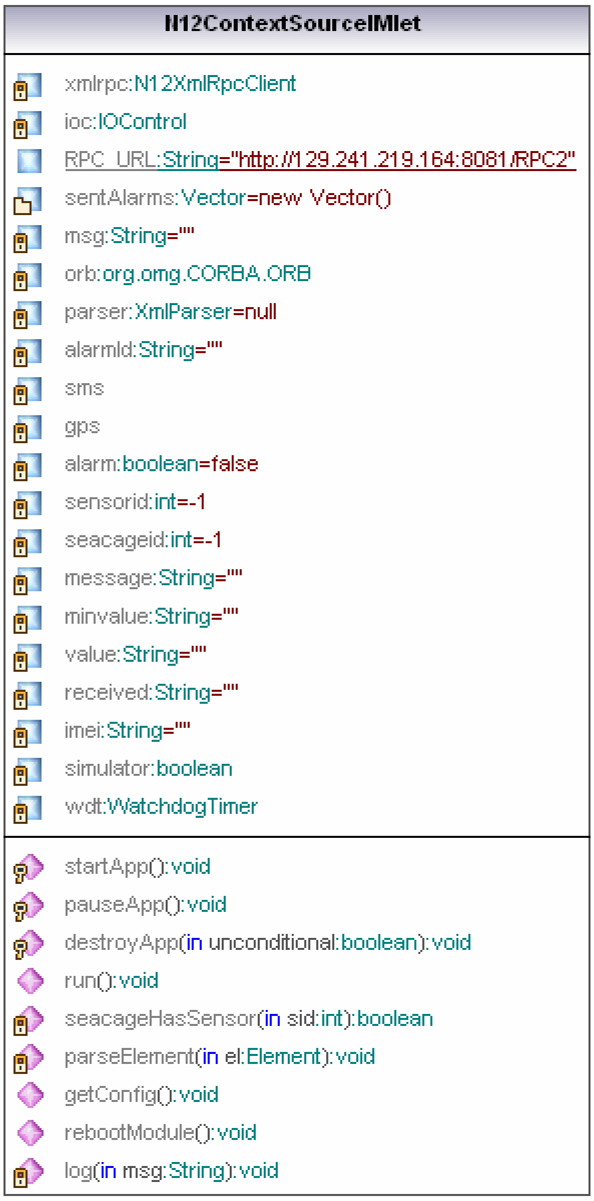


Figure 70: com.telenor.apms.fifamos.n12 class diagram

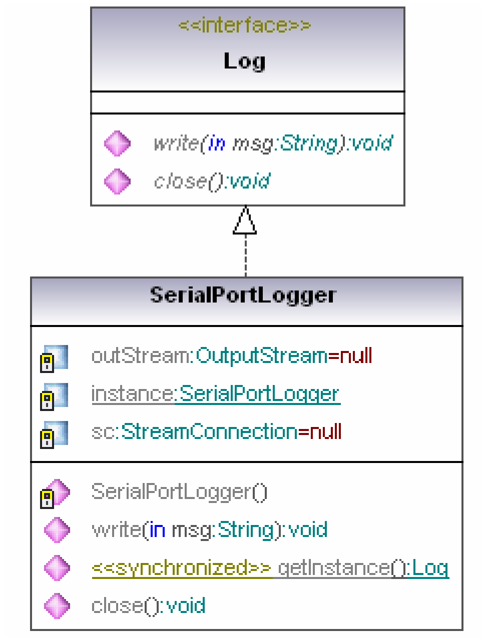

<<interface>>
Log
write(in msg:String):void
close():void
SerialPortLogger
outStream:OutputStream=null
instance:SerialPortLogger
sc:StreamConnection=null
SerialPortLogger()
write(in msg:String):void
<<synchronized>> getInstance():Log
close():void

Figure 71: com.telenor.apms.fifamos.n12.logger class diagram

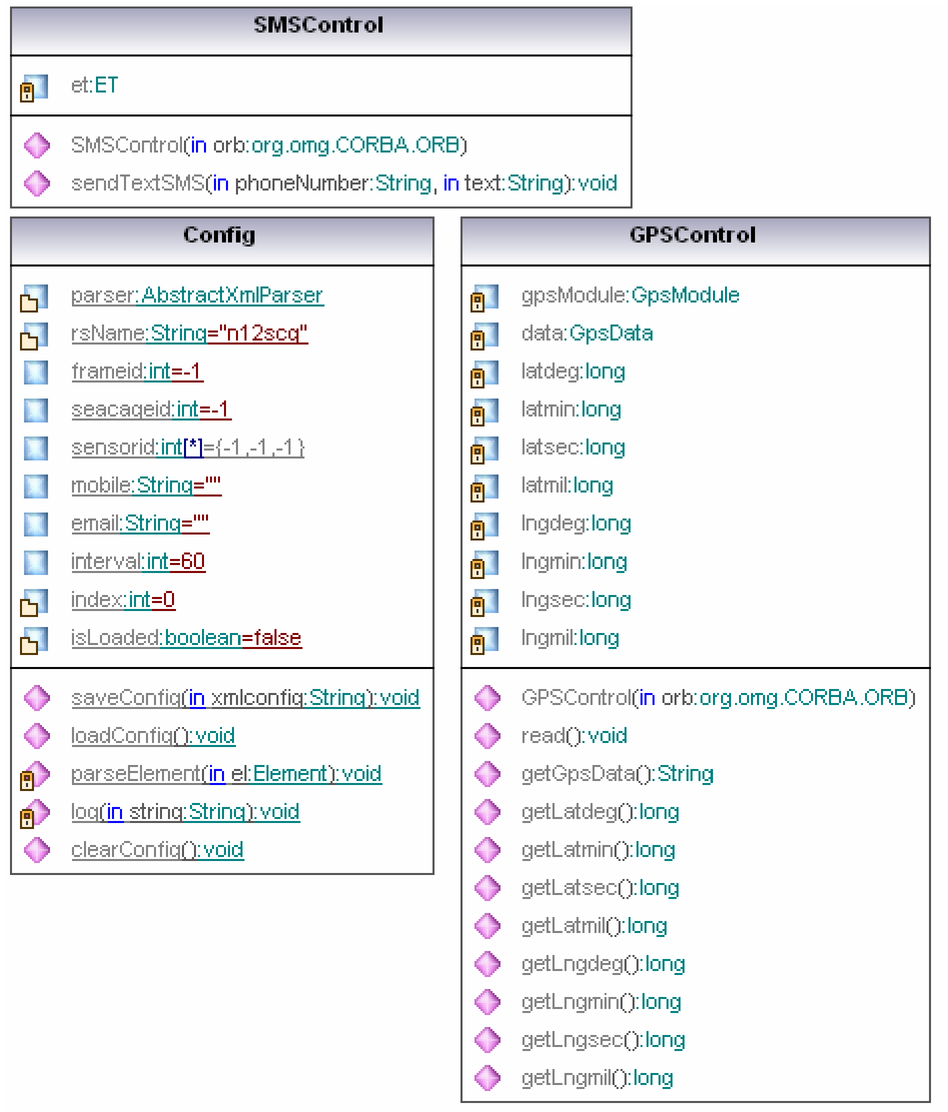


Figure 72: com.telenor.apms.fifamos.n12.utils class diagram

F.4 FiFaMoS Context Consumer

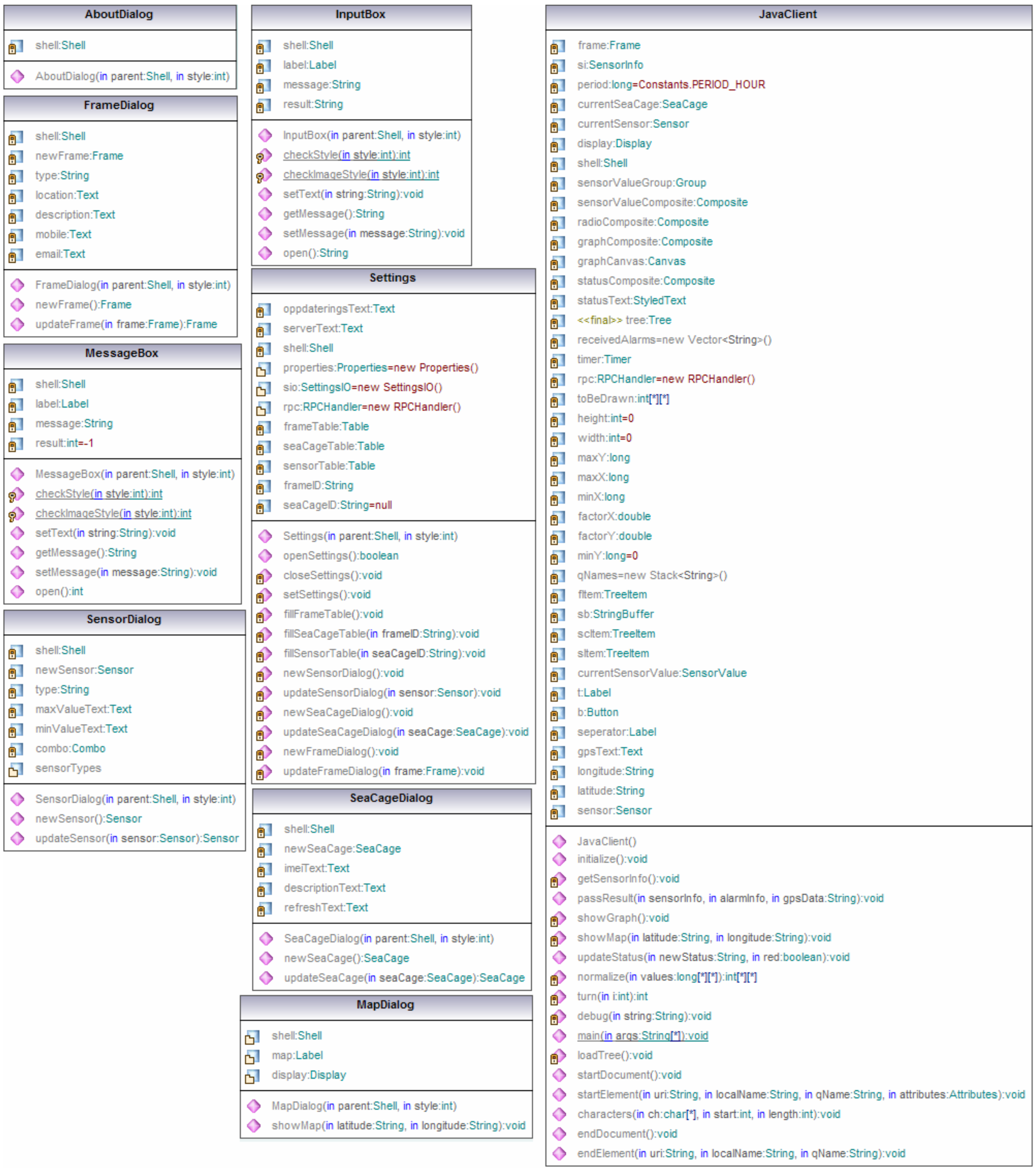
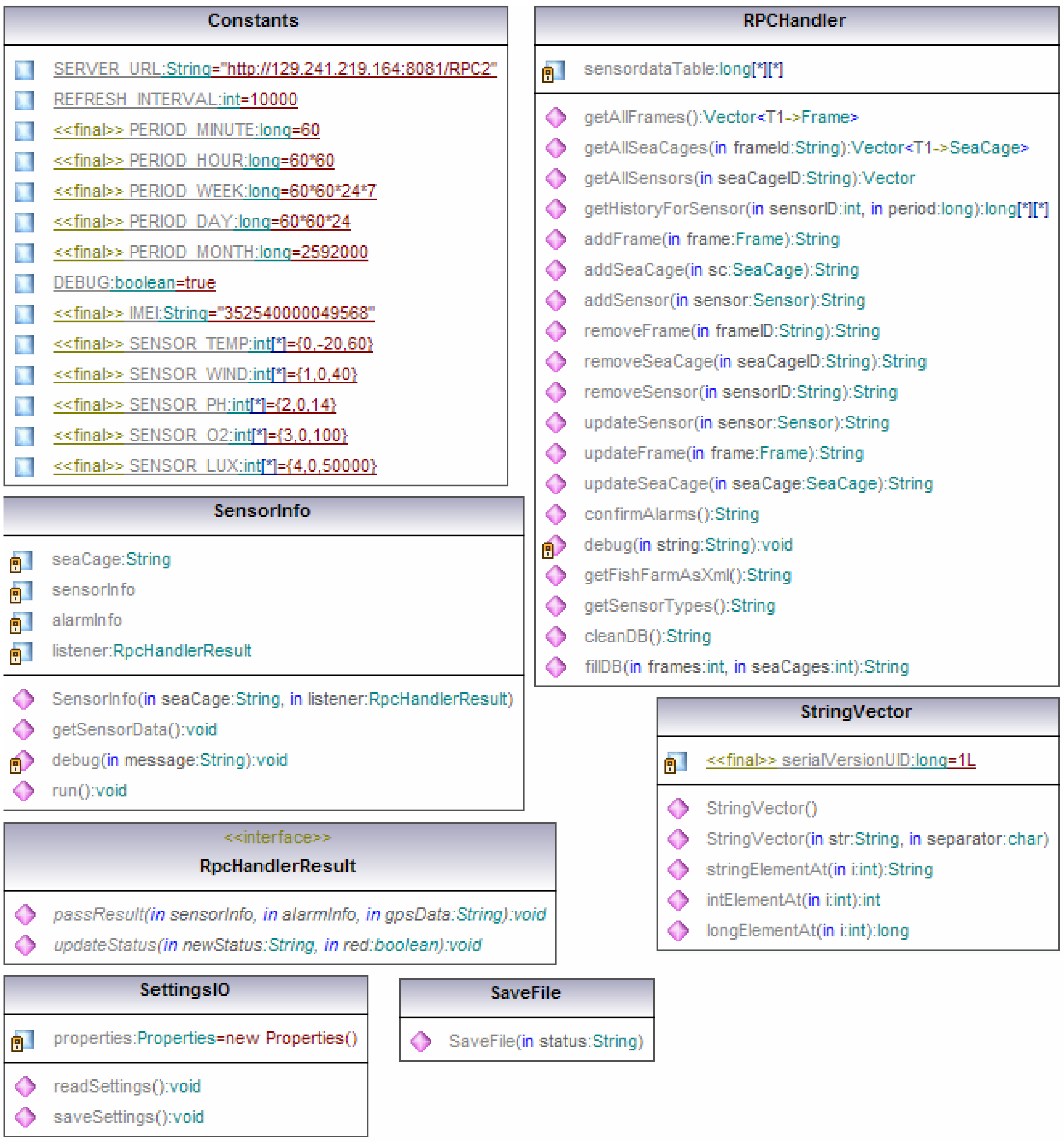

AboutDialog
shell:Shell
AboutDialog(in parent:Shell, in style:int)
FrameDialog
shell:Shell
newFrame:Frame
type:String
location:Text
description:Text
mobile:Text
email:Text
FrameDialog(in parent:Shell, in style:int)
newFrame():Frame
updateFrame(in frame:Frame):Frame
MessageBox
shell:Shell
label:Label
message:String
result:int=-1
MessageBox(in parent:Shell, in style:int)
checkStyle(in style:int):int
checkImageStyle(in style:int):int
setText(in string:String):void
getMessage():String
setMessage(in message:String):void
open():int
SensorDialog
shell:Shell
newSensor:Sensor
type:String
maxValueText:Text
minValueText:Text
combo:Combo
sensorTypes
SensorDialog(in parent:Shell, in style:int)
newSensor():Sensor
updateSensor(in sensor:Sensor):Sensor
InputBox
shell:Shell
label:Label
message:String
result:String
InputBox(in parent:Shell, in style:int)
checkStyle(in style:int):int
checkImageStyle(in style:int):int
setText(in string:String):void
getMessage():String
setMessage(in message:String):void
open():String
Settings
oppdateringsText:Text
serverText:Text
shell:Shell
properties:Properties=new Properties()
sio:SettingsIO=new SettingsIO()
rpc:RPCHandler=new RPCHandler()
frameTable:Table
seaCageTable:Table
sensorTable:Table
frameID:String
seaCageID:String=null
Settings(in parent:Shell, in style:int)
openSettings():boolean
closeSettings():void
setSettings():void
fillFrameTable():void
fillSeaCageTable(in frameID:String):void
fillSensorTable(in seaCageID:String):void
newSensorDialog():void
updateSensorDialog(in sensor:Sensor):void
newSeaCageDialog():void
updateSeaCageDialog(in seaCage:SeaCage):void
newFrameDialog():void
updateFrameDialog(in frame:Frame):void
SeaCageDialog
shell:Shell
newSeaCage:SeaCage
imeiText:Text
descriptionText:Text
refreshText:Text
SeaCageDialog(in parent:Shell, in style:int)
newSeaCage():SeaCage
updateSeaCage(in seaCage:SeaCage):SeaCage
MapDialog
shell:Shell
map:Label
display:Display
MapDialog(in parent:Shell, in style:int)
showMap(in latitude:String, in longitude:String):void
JavaClient
frame:Frame
si:SensorInfo
period:long=Constants.PERIOD_HOUR
currentSeaCage:SeaCage
currentSensor:Sensor
display:Display
shell:Shell
sensorValueGroup:Group
sensorValueComposite:Composite
radioComposite:Composite
graphComposite:Composite
graphCanvas:Canvas
statusComposite:Composite
statusText:StyledText
<<final>> tree:Tree
receivedAlarms=new Vector<String>()
timer:Timer
rpc:RPCHandler=new RPCHandler()
toBeDrawn:int[*][*]
height:int=0
width:int=0
maxY:long
maxX:long
minX:long
factorX:double
factorY:double
minY:long=0
qNames=new Stack<String>()
fItem:TreeItem
sb:StringBuffer
scItem:TreeItem
sItem:TreeItem
currentSensorValue:SensorValue
t:Label
b:Button
seperator:Label
gpsText:Text
longitude:String
latitude:String
sensor:Sensor
JavaClient()
initialize():void
getSensorInfo():void
passResult(in sensorInfo, in alarmInfo, in gpsData:String):void
showGraph():void
showMap(in latitude:String, in longitude:String):void
updateStatus(in newStatus:String, in red:boolean):void
normalize(in values:long[*][*]):int[*][*]
turn(in i:int):int
debug(in string:String):void
main(in args:String[*]):void
loadTree():void
startDocument():void
startElement(in uri:String, in localName:String, in qName:String, in attributes:Attributes):void
characters(in ch:char[*], in start:int, in length:int):void
endDocument():void
endElement(in uri:String, in localName:String, in qName:String):void

Figure 73: com.telenor.apms.fifamos.client.j2se.gui class diagram



```
Constants
SERVER_URL:String="http://129.241.219.164:8081/RPC2"
REFRESH_INTERVAL:int=10000
<<final>> PERIOD_MINUTE:long=60
<<final>> PERIOD_HOUR:long=60*60
<<final>> PERIOD_WEEK:long=60*60*24*7
<<final>> PERIOD_DAY:long=60*60*24
<<final>> PERIOD_MONTH:long=2592000
DEBUG:boolean=true
<<final>> IMEI:String="352540000049568"
<<final>> SENSOR_TEMP:int[*]={0,-20,60}
<<final>> SENSOR_WIND:int[*]={1,0,40}
<<final>> SENSOR_PH:int[*]={2,0,14}
<<final>> SENSOR_O2:int[*]={3,0,100}
<<final>> SENSOR_LUX:int[*]={4,0,50000}
```

```
SensorInfo
seaCage:String
sensorInfo
alarmInfo
listener:RpcHandlerResult
SensorInfo(in seaCage:String, in listener:RpcHandlerResult)
getSensorData():void
debug(in message:String):void
run():void
```

```
<<interface>>
RpcHandlerResult
passResult(in sensorInfo, in alarmInfo, in gpsData:String):void
updateStatus(in newStatus:String, in red:boolean):void
```

```
SettingsIO
properties:Properties=new Properties()
readSettings():void
saveSettings():void
```

```
SaveFile
SaveFile(in status:String)
```

```
RPCHandler
sensordataTable:long[*][*]
getAllFrames():Vector<T1->Frame>
getAllSeaCages(in frameId:String):Vector<T1->SeaCage>
getAllSensors(in seaCageID:String):Vector
getHistoryForSensor(in sensorID:int, in period:long):long[*][*]
addFrame(in frame:Frame):String
addSeaCage(in sc:SeaCage):String
addSensor(in sensor:Sensor):String
removeFrame(in frameID:String):String
removeSeaCage(in seaCageID:String):String
removeSensor(in sensorID:String):String
updateSensor(in sensor:Sensor):String
updateFrame(in frame:Frame):String
updateSeaCage(in seaCage:SeaCage):String
confirmAlarms():String
debug(in string:String):void
getFishFarmAsXml():String
getSensorTypes():String
cleanDB():String
fillDB(in frames:int, in seaCages:int):String
```

```
StringVector
<<final>> serialVersionUID:long=1L
StringVector()
StringVector(in str:String, in separator:char)
stringElementAt(in i:int):String
intElementAt(in i:int):int
longElementAt(in i:int):long
```

Figure 74: com.telenor.apms.fifamos.client.j2se.utils class diagram

F.5 FiFaMoS Mobile Context Consumer

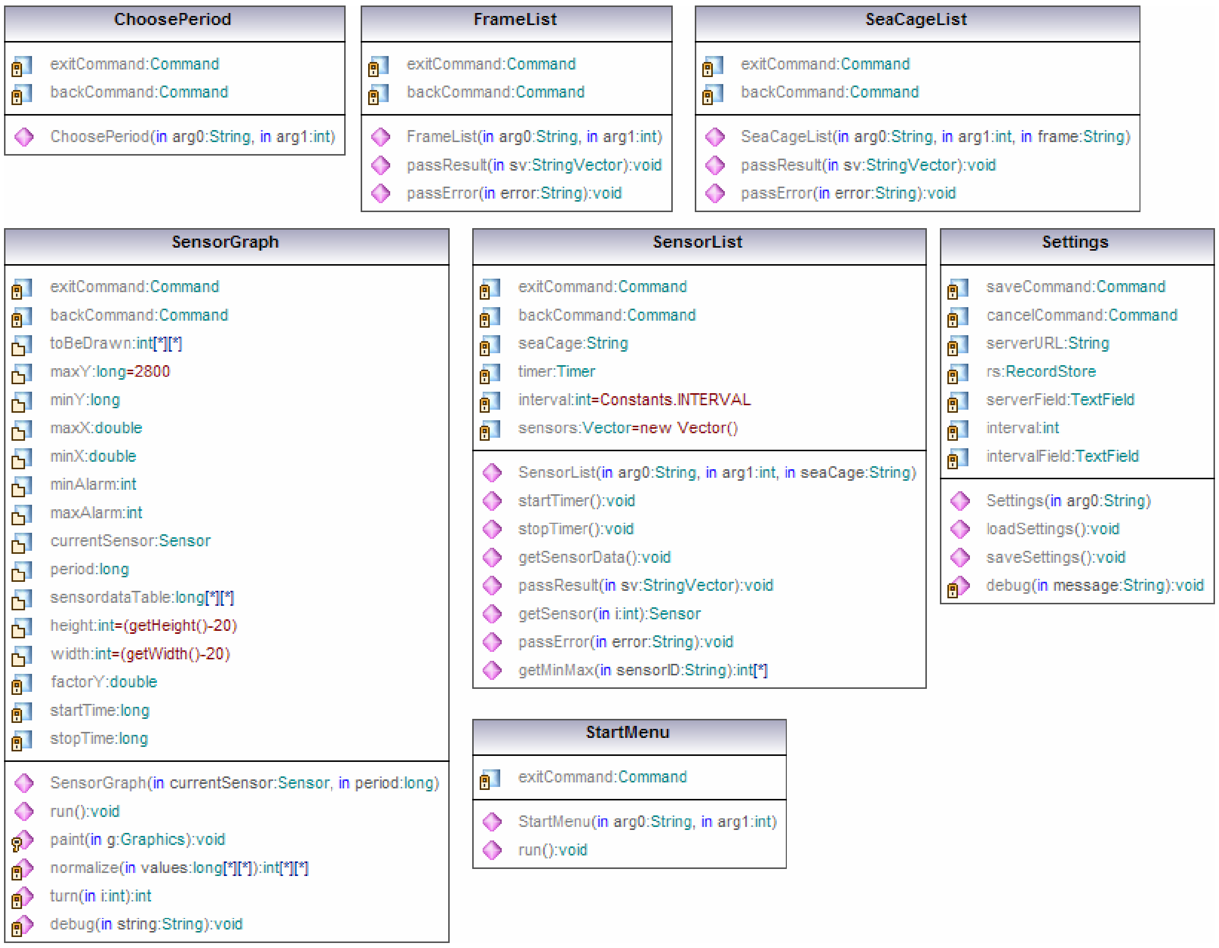

ChoosePeriod
exitCommand:Command
backCommand:Command
ChoosePeriod(in arg0:String, in arg1:int)
FrameList
exitCommand:Command
backCommand:Command
FrameList(in arg0:String, in arg1:int)
passResult(in sv:StringVector):void
passError(in error:String):void
SeaCageList
exitCommand:Command
backCommand:Command
SeaCageList(in arg0:String, in arg1:int, in frame:String)
passResult(in sv:StringVector):void
passError(in error:String):void
SensorGraph
exitCommand:Command
backCommand:Command
toBeDrawn:int[*][*]
maxY:long=2800
minY:long
maxX:double
minX:double
minAlarm:int
maxAlarm:int
currentSensor:Sensor
period:long
sensordataTable:long[*][*]
height:int=(getHeight()-20)
width:int=(getWidth()-20)
factorY:double
startTime:long
stopTime:long
SensorGraph(in currentSensor:Sensor, in period:long)
run():void
paint(in g:Graphics):void
normalize(in values:long[*][*]):int[*][*]
turn(in i:int):int
debug(in string:String):void
SensorList
exitCommand:Command
backCommand:Command
seaCage:String
timer:Timer
interval:int=Constants.INTERVAL
sensors:Vector=new Vector()
SensorList(in arg0:String, in arg1:int, in seaCage:String)
startTimer():void
stopTimer():void
getSensorData():void
passResult(in sv:StringVector):void
getSensor(in i:int):Sensor
passError(in error:String):void
getMinMax(in sensorID:String):int[*]
Settings
saveCommand:Command
cancelCommand:Command
serverURL:String
rs:RecordStore
serverField:TextField
interval:int
intervalField:TextField
Settings(in arg0:String)
loadSettings():void
saveSettings():void
debug(in message:String):void
StartMenu
exitCommand:Command
StartMenu(in arg0:String, in arg1:int)
run():void

Figure 75: com.telenor.apms.fifamos.client.j2me.gui class diagram

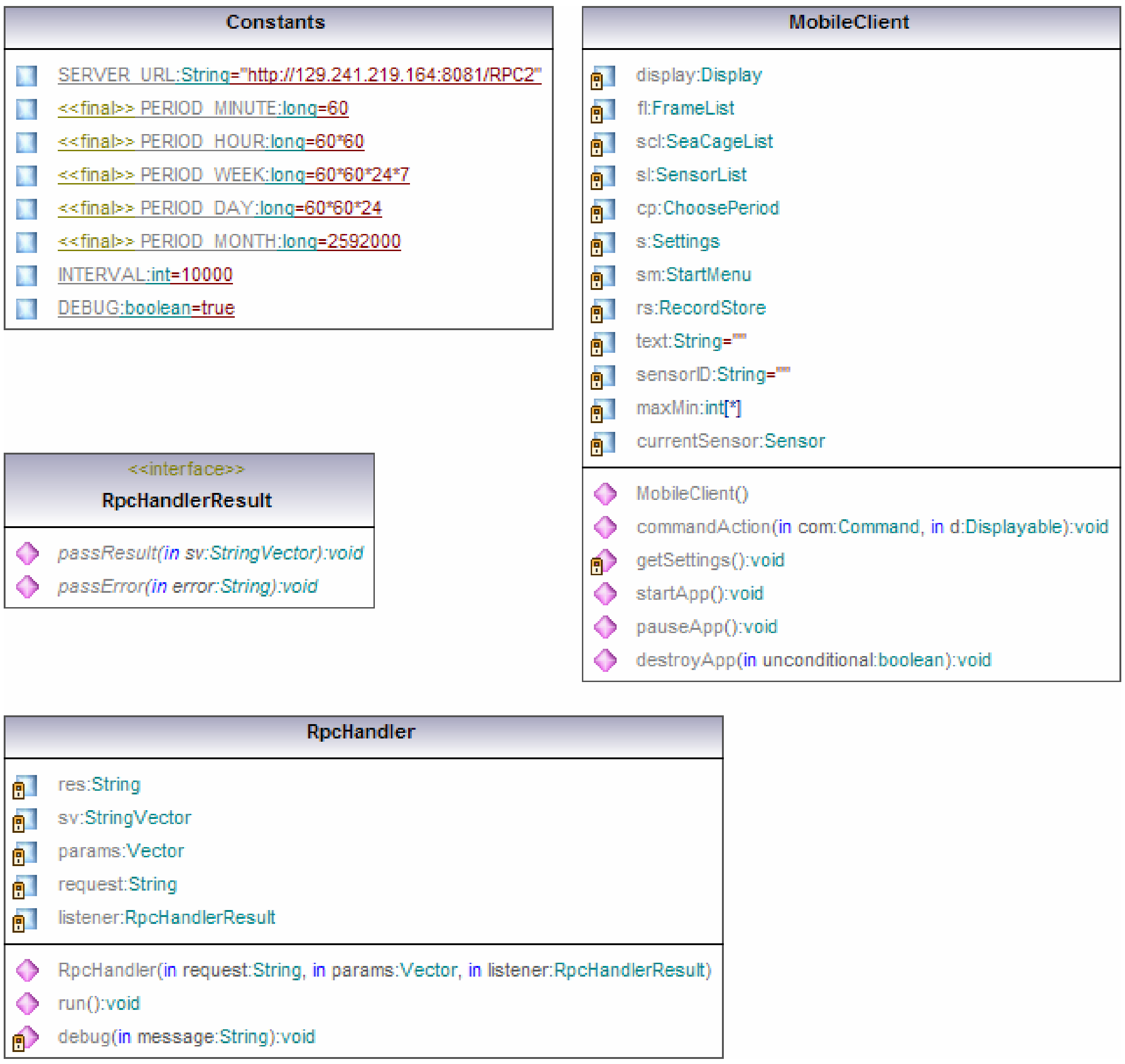

Constants
SERVER_URL:String="http://129.241.219.164:8081/RPC2"
<<final>> PERIOD_MINUTE:long=60
<<final>> PERIOD_HOUR:long=60*60
<<final>> PERIOD_WEEK:long=60*60*24*7
<<final>> PERIOD_DAY:long=60*60*24
<<final>> PERIOD_MONTH:long=2592000
INTERVAL:int=10000
DEBUG:boolean=true
MobileClient
display:Display
fl:FrameList
scl:SeaCageList
sl:SensorList
cp:ChoosePeriod
s:Settings
sm:StartMenu
rs:RecordStore
text:String=""
sensorID:String=""
maxMin:int[*]
currentSensor:Sensor
MobileClient()
commandAction(in com:Command, in d:Displayable):void
getSettings():void
startApp():void
pauseApp():void
destroyApp(in unconditional:boolean):void
<<interface>>
RpcHandlerResult
passResult(in sv:StringVector):void
passError(in error:String):void
RpcHandler
res:String
sv:StringVector
params:Vector
request:String
listener:RpcHandlerResult
RpcHandler(in request:String, in params:Vector, in listener:RpcHandlerResult)
run():void
debug(in message:String):void

Figure 76: com.telenor.apms.fifamos.client.j2me.control class diagram

Appendix G Description of the RUP process

The inception phase - In this phase the business case which includes business context, success factors (expected revenue, market recognition, etc), and financial forecast is established. To complement the business case, a basic use case model, project plan, initial risk assessment and project description (the core project requirements, constraints and key features) are generated.

This phase is documented in the prestudy chapter, in addition to the use cases stated in 3.2 Requirement specifications. No financial analysis of the business case is performed, due to the limitations of the thesis.

The elaboration phase - The elaboration phase is where the project starts to take shape. In this phase the problem domain analysis is made and the architecture of the project gets its basic form.

This phase was carried out as described, and is documented in 3.2 Requirement specifications and 3.3 Design.

The construction phase - In this phase the main focus goes to the development of components and other features of the system being designed. This is the phase when the bulk of the coding takes place.

This phase was completed as described, and is documented in chapter 3.4

The transition phase - In the transition phase, the product has moved from the development organization to the end user. The activities of this phase include training of the end users and maintainers and beta testing of the system to validate it against the end users' expectations. The product is also checked against the quality level set in the Inception phase. If it does not meet this level, or the standards of the end users, the entire cycle in this phase begins again.

The FiFaMoS system has not been tested by end users, but extensive testing has been completed of the developers, and the results are stated in 3.4.5 Testing.