Context-Aware Goods
Combining RFID Tracking and Environment Sensing

Sigve Albretsen
Mikael André Larsen

Master of Science in Computer Science
Submission date: June 2008
Supervisor: John Krogstie, IDI
Co-supervisor: Carl-Fredrik Sørensen, SINTEF

Norwegian University of Science and Technology
Department of Computer and Information Science

Problem Description

Item tracking and context information is becoming more and more important in the effort to ensure safe food and good food quality, especially for fresh food. Standards for item tracking and sensor information are emerging, but little work has been done to combine them.

This thesis will try to explore the field of context-aware information and tracking of goods. The work will partly be a continuation of the project EPC Information Services: Usage and Value Scenarios of EPCIS, completed fall 2007. This thesis will use EPCIS for tracking of goods, and investigate if SensorML can be used together with EPCIS for context-aware information.

The goal for the thesis is to develop an example software architecture and present a set of scenarios describing future uses. The software architecture shall combine an RFID tracking system with context information retrieved from sensors.

Assignment given: 15. January 2008
Supervisor: John Krogstie, IDI

ABSTRACT

Technology is becoming increasingly important in the effort to ensure safe food and good food quality, especially in the fresh food industry. Examples of such technology are systems for tracking and tracing food products, and the use of sensors to obtain context information about the environment. This technology is becoming more mature, and various standards are starting to emerge, but little work has been done combining these technologies or respective standards.

This thesis presents an example software architecture combining an RFID tracking system with context information retrieved from sensors. The sensors can be located both on the RFID tag itself and in locations where the items are, or have been, located. Two frameworks are combined in this architecture; EPC Architecture Framework for item tracking and Sensor Web Enablement for sensor and context information.

A set of scenarios describing potential uses of this technology is also presented. They are grouped by topics, with categories such as quality deterioration, temperature profiles, sensor collaboration and intelligent goods, hierarchy of goods with sensors, and proximity control. Each scenario is independent of the technical solution used, and does not require our architecture. The focus is on what can be achieved when a context-enabled tracking solution is implemented. These scenarios form the basis for the requirements specification of the architecture.

The thesis shows that the integration of the two standard frameworks can be achieved with relatively small modifications, and that the technology needed to achieve what is presented in the scenarios is already available. It is, however, necessary to perform pilot implementations and testing in order to find how best to utilize the technology.

PREFACE

This report presents our Master's Thesis in Computer Science at the Department of Computer and Information Science (IDI) at the Norwegian University of Science and Technology (NTNU). The work took place during the spring of 2008, and was conducted in cooperation with SINTEF Fisheries and Aquaculture.

We would like to thank our supervisors, Dr. Carl-Fredrik Sørensen from SINTEF and Professor John Krogstie from NTNU, for all their help and advice. They have provided valuable input and ideas for our work, as well as feedback on the final report. We would also like to thank Aleksander Lie for proofreading our thesis.

Trondheim, June 10, 2008

Sigve Albretsen

Mikael André Larsen

TABLE OF CONTENTS

Part I Introduction 1

1 Introduction 3

1.1 Motivation 3

1.2 Scope 3

1.3 Goals....4

1.4 Thesis Outline 4

2 Research 5

2.1 Research Context 5

2.2 Research Questions 5

2.3 Research Approach 5

2.4 Contributions 7

Part II Background and State of the Art 9

3 Introduction 11

4 Context-Aware Computing 13

4.1 Context 13

4.2 Context-Awareness....14

4.3 Context Applied to this Thesis 14

5 RFID Technology 17

5.1 Tag Characteristics 17

5.1.1 Active and Passive Tags....17

5.1.2 Powering a Passive Tag 18

5.1.3 Reusable Tags 19

5.1.4 Static or Dynamic Data....19

5.2 Challenges 20

5.2.1 Collocated Tags....20

5.2.2 Reliability....20

5.2.3 Reader Collision Problem 21

5.2.4 Cost 21

5.2.5 Security and Privacy 21

5.3 Standards....23

6 Sensors 25
6.1 Wireless Sensor Networks 25
6.1.1 Applications 26
6.1.2 Challenges 27
6.2 Different Types of Electronic Sensors 28
6.2.1 Manufacturers 29
6.2.2 Product Types 29
6.3 Chemical Sensors 31
7 Integrating RFID and Sensor Systems with EPC/EPCIS 33
7.1 Including User Data with Standard EPC Tags 34
7.2 Integrating with EPCIS 35
7.2.1 Using EPCIS Independent of Tag Type 36
7.2.2 Extending EPCIS to Fit Different Needs 37
7.2.3 EPCIS Interfaces 40
8 Administration of RFID and Sensor Networks 43
9 Sensor Web Enablement Standards 47
9.1 Observations and Measurements (O&M) 49
9.2 Sensor Model Language (SensorML) 50
9.3 Sensor Observation Service (SOS) 54
9.4 Sensor Planning Service (SPS) 56
9.5 Other Standards 58

Part III Own Contribution 61
10 Introduction 63
11 Evaluation and Comparison of EPC AF and SWE 65
11.1 EPC Architecture Framework 65
11.2 Sensor Web Enablement 67
11.3 Integration of EPC AF and SWE 70
11.4 Comparison of EPC AF and SWE 71
12 Stakeholder Analysis 73
12.1 Government 73
12.2 Consumers 73
12.3 Product Manufacturers 73
12.4 Transportation Companies 74

12.5 Retailers 74
12.6 Developers 74
13 Scenarios 75
13.1 Quality Deterioration (SC 1) 75
13.1.1 Automatic and Dynamic Expiration Dates (SC 1.1) 75
13.1.2 Reloading of Shipment Based on Quality (SC 1.2) 76
13.1.3 Proving the Quality of Fresh Food (SC 1.3) 77
13.2 Temperature Profiles (SC 2) 77
13.2.1 How Many Sensors Are Needed In a Given Space (SC 2.1) 77
13.2.2 Detection of Cold Spots and Warm Spots (SC 2.2) 77
13.2.3 Mapping the Need for Insulation of Goods Under Transportation (SC 2.3) 78
13.3 Sensor Collaboration and Intelligent Goods (SC 3) 78
13.3.1 Automatic Alarms of Violations (SC 3.1) 78
13.3.2 Automatic Environment Control (SC 3.2) 79
13.3.3 Collaboration of Goods (SC 3.3) 79
13.4 Hierarchy of Goods with Sensors (SC 4) 79
13.4.1 Automatic Aggregation of Goods with Sensors in a Hierarchy (SC 4.1) 80
13.5 Proximity Control (SC 5) 80
13.5.1 Fresh Food (SC 5.1) 80
13.5.2 Dangerous Goods (SC 5.2) 81
14 Requirements Specification 83
14.1 Scope 83
14.2 Functional Requirements 84
14.3 Requirements Relation to Scenarios and Research Questions 87
14.4 Quality Attributes 88
14.4.1 Security 88
14.4.2 Privacy 89
14.4.3 Scalability 89
14.4.4 Modifiability 89
14.4.5 Performance 90
15 Architectural Solution 91

15.1 Architectural Drivers 91
15.2 Choosing Viewpoints 91
15.2.1 What Are Viewpoints and Why Investigate Them 91
15.2.2 Evaluation of the Different Viewpoints 93
15.3 Quality Tactics 95
15.3.1 Modifiability 95
15.3.2 Performance 96
15.3.3 Security 96
15.3.4 Privacy 96
15.3.5 Scalability 97
15.4 Patterns 97
15.4.1 Separation of Concerns 97
15.4.2 Facade 98
15.5 Views 98
15.5.1 Logical View 99
15.5.2 Process View 106
15.5.3 Development View 110
15.6 Scenarios 114
15.6.1 Proving the Quality of Fresh Food (SC 1.3) 114
15.6.2 Automatic Alarms of Violations (SC 3.1) 116
15.7 Software Architecture Rationale 119
15.7.1 Security and Privacy 119
15.7.2 Scalability and Performance 119
15.7.3 Modifiability 119

Part IV Evaluation and Conclusion 121
16 Evaluation 123
16.1 Architectural Evaluation 123
16.1.1 Separation of Concerns 123
16.1.2 Scenario Evaluation 124
16.1.3 Requirements Evaluation 127
16.2 Research Questions Evaluation 130

16.3 Discussion 132
17 Conclusions and Future Work 135
17.1 Conclusions 135
17.2 Future work...135

PART V Bibliography and Appendices 137
Bibliography 139
A Acronyms...145
B Introduction to EPC Information Services 147

TABLE OF FIGURES

Figure 1 Research approach....6
Figure 2 Chemical sensor (TimeTemp 2008) 32
Figure 3 EPCglobal Architecture Framework (EPCglobal 2007c) 34
Figure 4 EPCIS layers (EPCglobal 2007a)....36
Figure 5 EPCIS Capture Interface (EPCglobal 2007a) 40
Figure 6 EPCIS Query Control Interface 41
Figure 7 EPCIS Query Callback Interface....41
Figure 8 Sensor Web example (Botts, Percivall et al. 2007) 47
Figure 9 Basic Observation type (Open Geospatial Consortium 2007a) 49
Figure 10 SensorML process model (Open Geospatial Consortium 2007c) 53
Figure 11 SensorML non-physical/physical processes (Open Geospatial Consortium 2007c).54
Figure 12 Consumer flow chart (Open Geospatial Consortium 2007e) 55
Figure 13 SPS overview (Botts, Percivall et al. 2007) 57
Figure 14 Registries in SWE (Botts, Percivall et al. 2007) 58
Figure 15 Deterioration of one type of food at two different temperatures....76
Figure 16 Overview of the environment 83
Figure 17 The 4+1 View Model (Kruchten 1995) 92
Figure 18 Separation of concerns....97
Figure 19 Facade pattern 98
Figure 20 Logical view....100
Figure 21 EPCIS parts of the system 101
Figure 22 Sensor Web Enablement parts of the system 103
Figure 23 Context-Aware Goods System parts 103
Figure 24 Context data flow diagram 106

Figure 25 Top level data flow diagram 107

Figure 26 Observation registration 109

Figure 27 Development view 111

Figure 28 The role component syntax..112

Figure 29 The interface component syntax 113

Figure 30 Proposed solution of SC 1.3 by CAGS..115

Figure 31 Proposed solution of SC 3.1 by CAGS; The management of sensors.....................116

Figure 32 Proposed solution of SC 3.1 by CAGS; The subscription 117

Figure 33 Proposed solution of SC 3.1 by CAGS; The alert 118

Figure 34 EPCglobal Architecture Framework (EPCglobal 2007c) 149

Figure 35 EPCIS layers (EPCglobal 2007a)..150

Figure 36 EPCIS event types (EPCglobal 2007a)..154

Figure 37 Service Layer interfaces (EPCglobal 2007a)..156

Figure 38 Comparison of ONS and EPCIS Discovery (University of Cambridge, AT4 wireless et al. 2007) 160

TABLE OF TABLES

Table 1 Temperature sensors 30
Table 2 Acceleration sensors 31
Table 3 Humidity sensors....................31
Table 4 EPCIS data extension methods (EPCglobal 2007a) 39
Table 5 O&M example 50
Table 6 Functional requirement to research question....................87
Table 7 Functional requirement to scenarios....................88

Part I Introduction

1 INTRODUCTION

This chapter will briefly explain the motivation for our research, our scope, and present the different parts of the thesis, giving a brief overview of the structure of the report. The next chapter presents our research questions, research context, and the research methods used. It will also present our main contributions.

1.1 MOTIVATION

The need for context information concerning tangible products in a supply chain is increasing. Laws in the EU and Norway will demand tracing and control of products, and a big issue regarding control is the application of context information. Fresh food have to be transported at given temperatures, other goods like electronics have to be handled gently. By tracking context information about a product, it can be proved when a product has been treated as it should, and when it has not. This information is necessary for internal use, but has great potential when shared with others within a value chain.

The technology behind RFID (Radio Frequency Identification) has been around for quite a long time, but it is only in latter years that it has been used at a larger scale. Different industries have shown interest in this technology, where the benefits of identifying items on an individual level are obvious. One industry that has been very interested is the retail industry, which handles large amounts of items and has complex logistics that could be supported by RFID. RFID can be used to track purchases, do inventory management and control, and automate manual processes such as dispatch and receipt of goods.

In many cases the tracking of an identity is not enough, and additional information is needed. This is especially relevant in the fresh food industry where information about, e.g., temperature of the goods is very important. It is interesting to know where the food is coming from, where it has traveled, and how it has been treated on the way. Some retail chains use this information today as an extra service to customers, but it is likely that in a few years this will be mandatory for all companies dealing with fresh food. Both the Norwegian government and the European Union have projects on tracing fresh food.

1.2 SCOPE

There are many possible angles of approach to a problem such as this. It is possible to look at, e.g., hardware requirements, communication between sensors (smart sensors), interoperability between solutions from different vendors or industries, and how to share the information generated. Our focus is on the software level, and we will look at the situation from after the information from the sensors and tags have been read and entered into the software systems. We will elaborate about the technology behind RFID and different sensor types in the background section, but this is only to understand the situation and will not be researched further. Our focus is not on the hardware level, and we will assume that this is working. We will not look at filtering tag reads, false reads, read reliability and other such issues. We assume that the information received is correct, and we focus on how this information can be used.

Our interest lies in combining information from sensors and RFID tags, potential uses of such information, and how this can be shared between different organizations. What we find interesting is how software architectures can support generic information from sensors and RFIDs, and other relevant information. It should also be easy to add new sensor types and different behaviors and actions based on the information available.

1.3 GOALS

The main goal of this project is to develop a software architecture combining RFID tracking with sensor information. The architecture will be generic with regard to sensor types and sensor data supported. The secondary goals are to investigate and evaluate two standard frameworks which will be used in our architecture, Electronic Product Code Architecture Framework (EPC AF)(EPCglobal 2007c) and Software Web Enablement (SWE)(Botts, Percivall et al. 2007), as well as present scenarios for potential uses of such a solution.

1.4 THESIS OUTLINE

This thesis is divided into five parts: Introduction, Background and State-of-the-Art, Own Contribution, Evaluation and Conclusion, and Bibliography and Appendices.

- **Part I: Introduction** presents the background and motivation for the thesis, how we have proceeded in our work, what we have achieved, and present the rest of the report.
- **Part II: Background and State-of-the-Art** presents the technology behind RFID, different sensors, issues to consider related to administration of large RFID/sensor networks, and how EPC and EPCIS can be used for RFID and sensor networks. It also presents the Sensor Web Enablement framework for sensor webs.
- **Part III: Own Contribution** presents our contribution. It presents an evaluation of two standard frameworks, scenarios for how the technology can be used, as well as a requirements specification and architecture for a software solution supporting the scenarios.
- **Part IV: Evaluation and Conclusion** evaluates our contribution. It also concludes the report and defines future work.
- **Part V: Bibliography & Appendices** presents the references used, acronyms, and an introduction to EPC Information Services (EPCIS).

2 RESEARCH

This chapter will present the context for our research, our research questions, and the approach used to answer these. It will also briefly present our contributions.

2.1 RESEARCH CONTEXT

This thesis is partly a continuation of a project done last fall in cooperation with Accenture (Albretsen and Larsen 2007), a project which focused on the Electronic Product Code Information Services (EPCIS) standard from EPCglobal, and potential uses of this. The original problem description for this thesis was proposed by Accenture.

The problem description has been further developed in cooperation with SINTEF Fisheries and Aquaculture, which among other things is conducting research on the tracking of fresh fish and related context information. Related projects at SINTEF are “Smart Vareflyt”, SILO, InnovaRFID, and Seafood Plus. Important research areas are traceability and documentation, and food safety. The work on this thesis has been conducted in close cooperation with SINTEF.

2.2 RESEARCH QUESTIONS

These are the research questions investigated in this thesis:

RQ 1. How can an electronic ID tracking information system be extended with sensor information?
RQ 2. How to achieve relative positioning of one tagged unit with regards to other tagged units?
RQ 3. How to build a generic software architecture towards sensors and tags that can be used in many different situations?
RQ 4. How to ensure scalability in an ID tracking information system extended with sensor information?
RQ 5. How can a system with sensor information be integrated with EPC and EPCIS from EPCglobal?
RQ 6. How can SensorML be integrated with EPCIS?

2.3 RESEARCH APPROACH

The research approach has consisted of five steps, as shown in Figure 1. First an extensive literature study on the state of the art of the relevant research fields was conducted. The fields investigated were related to the problem description and included context and context-aware computing, RFID technology, sensors and sensor networks, related standards etc. This is, as mentioned above, a continuation of a previous work that included a literature study on silent commerce, the EPC Architecture Framework, and the EPC Information Services.

An evaluation of relevant standard frameworks was conducted after the literature study. The frameworks evaluated were the EPC Architecture Framework and Software Web Enablement. These were evaluated with regards to their usefulness for context information and item tracking. In parallel a set of application scenarios were described, where the possibilities of using this technology are described. This could be done in parallel as the scenarios describe the possibilities of combining item tracking with context information without depending on specific technology or solutions.

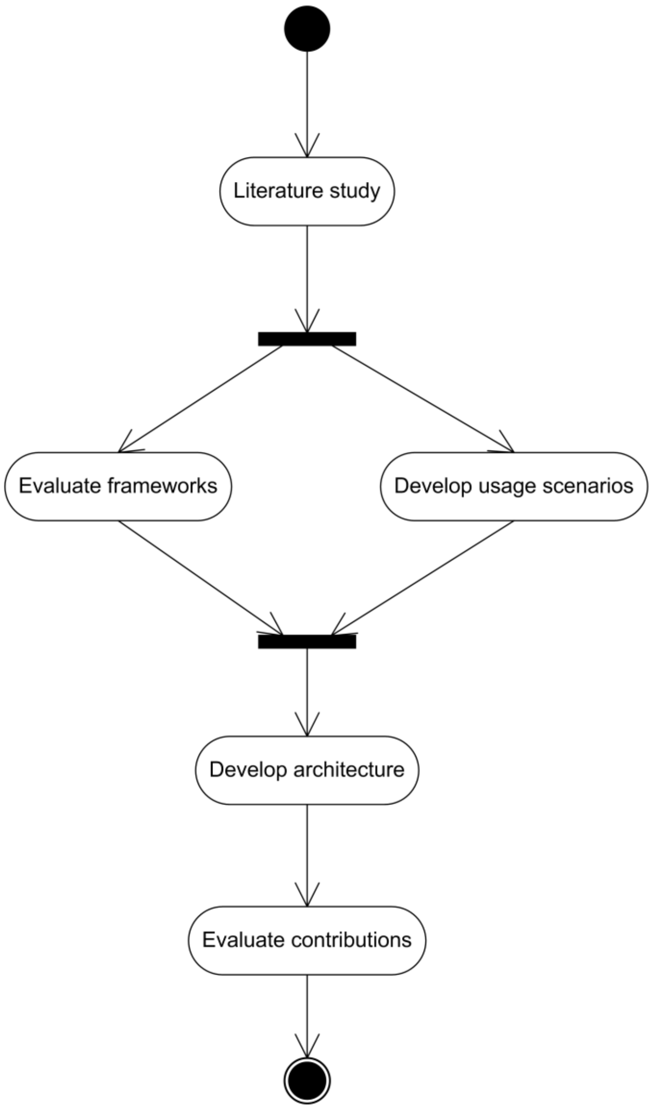


Figure 1 Research approach

The next step was to develop a software architecture which can be used when creating applications enabling these scenarios. The frameworks evaluated in the previous step were integrated into this architecture. Finally, the solution was evaluated with regards to the research questions stated above.

2.4 CONTRIBUTIONS

The main contribution of this thesis is a software architecture description of a solution utilizing standard frameworks from EPCglobal and OpenGIS to provide a context-enabled RFID tracking system. This architecture will serve as an example of how two such architectures can be integrated and how generic sensors and context information can be included in a tracking system. The architecture description includes information and specification of stakeholders, requirements, quality attributes, tactics, patterns, and views. Another important contribution is a set of scenarios presenting motivations for this architecture. The scenarios are grouped according to topic, with a description of what kind of data will be needed and how this information can be used. Each requirement in the architecture is tied to one or more scenarios. The thesis also gives an evaluation of the two standard frameworks used in the architecture with respect to the two areas item tracking and context information. Finally, the contributions are evaluated, and some conclusions and areas of future work are presented.

Part II Background and State of the Art

3 INTRODUCTION

This part describes the background for the thesis, and is the result of a literature study. This study was conducted to gain enough knowledge to perform the work done later in this thesis. The part introduces the technology behind RFID and issues related to RFID networks, and it also discusses different possibilities for sensors related to RFID networks. Finally, the Sensor Web Enablement framework for sensor networks is presented.

The goal of this part is to present topics necessary as background when reading the parts that follow later in the thesis. It is also the intention to give readers unfamiliar with RFID and wireless sensors an introduction to these areas.

OUTLINE

The rest of this part is divided into six chapters:

- **Chapter 4: "Context-aware computing"** defines what context is, and how applications can be context-aware. Context is then related to later chapters, and its meaning in RFID and sensor systems are presented.
- **Chapter 5:"RFID Technology"** presents the technology behind RFID. Different characteristics for RFID tags, challenges for RFID, and relevant standards are presented.
- **Chapter 6: "Sensors"** discusses wireless sensors and sensor networks in general, before a description of different types of wireless sensors. The focus is on sensors related to RFID and which sensor types are available.
- **Chapter 7: "Integrating RFID and sensor systems with EPC/EPCIS"** describes how EPC and EPCIS can be used in a setting where both RFID and sensors are part of the system. It presents the different possibilities to extend EPC/EPCIS with information from sensors.
- **Chapter 8: "Administration of RFID and sensor networks"** presents important issues to consider when implementing solutions based on RFID and wireless sensors.
- **Chapter 9: "Sensor Web Enablement Standards"** presents the standards in the Sensor Web Enablement work done at the Open Geospatial Consortium.

4 CONTEXT-AWARE COMPUTING

The use of context is important in many types of applications. It is especially important if the context changes frequently, and the application needs to reflect this. This chapter will present what context is, how applications can be context-aware, and how context relates to this research.

4.1 CONTEXT

There are many different definitions of context, and it is difficult to determine exactly what the term means. It is often used as a vague term, because it is used in so many different situations. Most people have a certain idea what it means, but it is difficult to describe accurate. Merriam-Webster's dictionary defines context as *"Interrelated conditions in which something exists or occurs"* (Merriam-Webster 2008). This is a very general definition, and it does not help in understanding the term related to computer science. People may think of context as something which defines the situation we are in, such as location, date, and time; but this is not a formal definition. A problem with definitions which lists other terms to describe context, such as location, date and time, and nearby people, is that it is difficult to determine if information other than those listed is context. Another way to define context is by using synonyms for context. Examples are the environment, situation, user's environment, application's environment, and situation of the user. This is according to (Dey and Abowd 1999) just as limiting as the example above, as it is difficult to apply in practice. Even though (Dey and Abowd 1999) says that context definitions based on examples is a bad idea, they present four types of context they consider more important than others: *location*, *identity*, *activity* and *time*.

(Chen and Kotz 2000) mentions four different categories for context, and two aspects. The categories are computing context, user context, physical context, and time context. Having these categories, and especially time, it will be possible to obtain a context *history*, which can be interesting in many situations. The two aspects are active and passive. Active context is context that influences the behavior of the application, while passive context is relevant but not critical context. Their definition of context is: *"the set of environmental states and settings that either determines an application's behavior or in which an application event occurs and is interesting to the user"*.

The definition above is only relevant for computer applications, and for this thesis something more broadly is needed. It must be possible to define the context of e.g. fresh food or other products tracked by RFID. Context will therefore for the rest of this thesis be defined as by (Dey and Abowd 1999):*"Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications* *themselves".* This definition also focuses on software applications, but it defines context related to an item relevant for the application, e.g. RFID-tagged fresh food.

4.2 CONTEXT-AWARENESS

Context-aware applications are in essence applications which know about context, and can use this in some way. (Chen and Kotz 2000) defines two kinds of context-awareness in the same way as they define context: active and passive. Active context-awareness is applications adapting to the context, and which changes behavior according to context. Passive context-awareness is applications that presents context information to a user, or stores it for later use. (Dey and Abowd 1999) defines context-awareness a little different: *"A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task"*. This is a very general definition, and fits about every situation, and it combines the active and passive definition above into one definition by saying that applications provide information (passive) or services (active) to the user using context. They also present a list of three context-aware features that may be supported by such applications: Presentation of information and services to a user, automatic execution of a service, and tagging of context to information for later retrieval.

Both these definitions have things in common, and they rely on two possible uses for context: adapting to context and presenting context. In addition, it is possible to store the context information for later use. When this stored information is used it will be for one of the two above; adapting to or presenting context. A typical example of a context-aware application presenting context is a tourist guide which uses the location of the user to present relevant information on the surrounding environment. E.g. at an art gallery the application can automatically present information on the closest painting. An application that adapts to context can be a print service on a PDA automatically sending print jobs to the nearest printer, which changes as the user moves through an office environment. A device presenting different functionality based on context, and not just changes functionality, is a cell phone which only allows outgoing calls when a network is in range. This is a common feature, but it could be possible for the phone to allow calls in all situations. The result when the network is out of reach would be that the user would not succeed in placing a call, and he would not know why.

4.3 CONTEXT APPLIED TO THIS THESIS

In this thesis, context is the identification and positioning of items, tracing of items, and sensor data related to items. Both the identification/tracing information and sensor data are relevant for presentation to users as well as for applications adapting to the information. Trace information can be used passively by presenting the steps a product has taken from the manufacturer to the store to a user. It can also be used actively to detect if an item has passed a boundary and raise an alarm, e.g. for theft prevention. This is the same for sensor data, it can be presented to a user who then decides if he wants to act on it, or it can be used more actively, e.g. by a refrigerator which automatically turns down the temperature if a product inside is too warm.

It is obvious that the easiest way to handle context information is to present it to a user, and then let the user decide what to do. If e.g. an item is located at several different positions, it is not difficult to plot these on a map to provide trace information. It requires a bit more logic for an application to take this information, process it, and then present to the user if the item has deviated from its planned route, in what state it is (e.g. if the last known location was the garbage), if it has been transported by air etc. The most advanced way to handle context information is applications which automatically act on this information. It can for example be possible for the application to predict when the item will arrive at a cross-docking site, and automatically order further transportation to be ready at that time.

Common for all these situations is that they have a tracking solution as a foundation, and build on this. This means that when such a solution is in place, it is possible to start with the easiest type of context-awareness, just presenting the information to a user, and later expand the solution with more advanced functionality.

In RFID-based systems, the identification of items is inherent. This thesis tries to get additional context information and relate this to the identified item. Using various sensors it is possible to get a lot of different context information which helps define the state of the item, the situation it is in, how it has been treated, where it has been, and when it was there. If all this information is combined and stored for future use, a very detailed history of an item’s lifetime can be obtained. Not only where the item was and when it was there, but also information on the state and situation the item was in at that time. Possibly could information on the item’s state between the locations also be obtained.

RFID is based on locating tags at “checkpoints” where there are RFID readers, and it gives no information on what happened between these readings. This information can be added by using sensors storing context information between tag-reads.

5 RFID TECHNOLOGY

This chapter gives an introduction to the technology behind RFID (Radio Frequency Identification), and looks at different characteristics and possibilities for RFID tags. Challenges when implementing RFID solutions are discussed, and some important standards related to RFID are presented.

RFID has been around for quite a while. It was e.g. used by the British military during World War II to identify aircrafts. Still it took many years before it gained a wide interest. Now there are much publicity regarding RFID, and many companies are looking at how RFID can benefit their business. RFID can be a natural replacement for the old barcode, and it has several benefits over barcodes. It can also be used as a supplement to barcodes with additional information and for redundancy. One important advantage of RFID is that it does not require a clear line of sight between the reader and the tag, which makes it possible to read larger quantities of tags at the same time. This thesis will not go into detail on the history of RFID, for more information, see (Landt 2005).

5.1 TAG CHARACTERISTICS

There are many different types of RFID tags using the same or similar technology, but which have different properties. It is therefore important to know what is available, and to choose the correct tag for a given situation. This section presents some characteristics or properties of RFID tags to give an overview of the possibilities of RFID.

5.1.1 ACTIVE AND PASSIVE TAGS

RFID tags are usually categorized into two different types; active and passive. The main difference between these is in how they are powered. Active tags are powered by their own power supply, either internally from a battery or connected by a power cord. Passive tags are without a power supply and receive power only when they are read. This means that passive tags only have power when in the range of a reader, and they are not capable of continuously collecting or processing data. For RFID used solely for identification, this is no problem as the tag will be powered when someone tries to read it, and will remain inactive at all other times. When the tags are required to process data at any time, e.g. when combined with a temperature sensor, an active tag is required.

Some monitoring is possible with passive tags as well. It is e.g. possible to monitor if a package has been opened by creating a small circuit around the entire package which will break when opened. This can be detected upon reading by submitting a single bit saying if the circuit leads power or not. This method enables only simple monitoring and an active tag is needed for more advanced functionality. The other possibility is to include a passive tag for identification, and have active sensors independent of the RFID tag.

Active tags have longer range than passive, as they can transmit with higher power. They are also capable of transmitting a beacon signal at even intervals which will be captured by readers whenever one is in range. The downside is that the battery has a limited lifetime, and that the price is higher than for a passive tag. It is also possible to have a semi-active tag. Semi-active tags have an onboard power source, but they do not try to transmit any signals before a given event occurs. Such tags could for example be used for alarm signals, where the tag gives notice if something happens but otherwise stays quiet.

There are also semi-passive tags. These tags are similar to active tags in the sense that they have their own power supply in the form of a battery, but this power is only used on the tag and not for transmitting a beacon signal or for broadcast. This results in a longer battery life than for active tags, and communication functions as for passive tags. The battery may be used to assist the receiving circuit on the tag which results in better reading range. The onboard power can for example be used to log environmental data, and it can therefore provide dynamically created data which is not possible for passive tags (Xiao, Yu et al. 2006) (Want 2006).

5.1.2 POWERING A PASSIVE TAG

There are two different approaches for powering a passive tag from a reader based on the electromagnetic properties near-field and far-field of an RF antenna. The near-field uses magnetic induction while the far-field uses electromagnetic wave capturing. The technology for sending the response from the tag to the reader is also different, near-field uses load modulation while far-field uses back scattering. Technical details will not be discussed here, for more information see (Want 2006).

It is one important problem by having two different approaches for passive RFID; they are not compatible. Near-field uses HF frequencies while far-field uses UHF. One important technology within near-field is NFC (Near Field Communication). NFC tries to enable wireless mobile devices to communicate with other devices on a short range without relying on discovery mechanisms used by other standards. The typical example of NFC is a mobile phone with a built in RFID tag and reader which can read passive RFID tags using the near-field approach. Potential uses for this are many, e.g. to use a cell phone as a means of payment. When entering a bus a reader at the door can detect the RFID tag in the cell phone, and payment can be done using the phone network such as GSM (Want 2006).

The first passive RFID tags were based on near-field, but there are some problems with this technology. The range for near-field is not too good due to limitations of magnetic induction, and it is reduced as the frequency is increased. The range for magnetic induction is approximately $\frac{c}{2\pi f}$, where c is the speed of light and f is the frequency. The frequency needs to be increased as the load is increased, and the range is thus reduced. Therefore tags based on far-field communication have been developed. The EPC (Electronic Product Code) from

EPCglobal is one example of the use of far-field RFID. Near-field usually operates at frequencies up to 100MHz, while far-field is above this level (Want 2006). The current EPC standard uses UHF frequencies (Ultra High Frequencies), but the standard mentions the possibility of a HF (High Frequency) tag using the near-field capabilities in the future. According to (Roberti 2007), this standard was supposed to be ratified in the middle of 2007, but as of June 2008 no standard has been released. When this standard has been ratified, it will be a great benefit to end users as they can choose between HF and UHF according to whatever fits their needs, and still conform to EPC standards.

5.1.3 REUSABLE TAGS

In some contexts reusable tags are important and are a necessity for implementing RFID. An example is Nortura, the leading supplier of meat and eggs in Norway. They have a special reusable container used for meat. Nortura has started an RFID project and is in the process of tagging each and every one of these containers. It has been very important for them that the tags are reusable and robust. It is for example necessary for the tags to endure washing at very high temperatures, as well as temperatures below freezing when containing frozen products. As these tags are cast inside the plastic of the case, they need to have a lifetime at least as long as the case itself (Logistikk & Ledelse 2007).

Another example where reusable tags are necessary is when using expensive tags with many functions. The cost difference between passive and active tags is very large, thus when using an active tag with added sensors or other functionality, reusability becomes important.

A situation where reusable tags are not an issue is when tagging consumer goods. Consumers will not want to return the tag when buying a product and the tag only needs to be usable until retail. Another example is when tagging disposable containers. The cost of removing the tag and applying it to a new container could be larger than the cost of the tag itself, depending on the type of tag used, and it would thus be thrown away with the container.

5.1.4 STATIC OR DYNAMIC DATA

Another characteristic of RFID tags is the data stored on the tag. Every tag has the ability to store and communicate static data, but some also has the ability to have dynamic data. Dynamic data is important when incorporating sensors on the tag to enable storing of data sampled from the sensor. Some tags have a kind of “semi-dynamic” memory in the sense that there is a user memory but it is writable only once. This can also be viewed as the difference between read-only and read-write memory. As always it is a question about cost and functionality, read-only is cheaper but read-write has better functionality. The possible uses of the user memory can both be predefined and optional to the implementer. As an example, the Tag Data Standard from EPCglobal clearly specifies the data types of the user memory. For added security, it is possible to have access control when reading or writing to the tag, typically by supplying a password with the read/write command (Want 2006) (EPCglobal 2007b).

5.2 CHALLENGES

There are many challenges to consider when using RFID. Many of these have been solved, but implementers need to be aware of them. This section will describe some of these.

5.2.1 COLLOCATED TAGS

It is problematic to read collocated tags, as there are problems with collision in transmission in all wireless communication. This needs to be considered when producing RFID readers, but for end users this problem has been solved. The solution used by e.g. the EPC Class 1 tags is a Query Tree protocol, which models the ID space as a binary tree and tries to read different parts of the tree at a time, until it only reads a node which corresponds to a tag identity. Similar solutions exist for other tag air interfaces.

5.2.2 RELIABILITY

Highly reliable reading has been a problem for RFID solutions, where less than 100% tag read success are in many cases not an option. It can e.g. be very expensive if a checkout counter at a store does not register every item that passes through. It is also necessary to have a different approach when trying to correctly identify one tag, e.g. for RFID-based access control, and when trying to correctly identify an unknown number of tags at the same time, e.g. at a supermarket (Vogt 2002). The material of the object tagged does also have an impact on the success of a tag read. It is important to consider the material of the tagged objects, and to choose tags accordingly, as it can interfere with the radio waves (Ukkonen, Engels et al. 2005). It is however possible to achieve high read rates for RFID and Pat King argues in (King 2006) that it is possible to achieve Six Sigma reliability with RFID. Six Sigma refers to the ability of a process to perform within specification. The goal is that there should be six standard deviations between the mean and the specification limits. According to (Michael and McCathie 2005), the San Francisco International Airport has consistent read rates above 99.5%, but, as mentioned above, in many cases 100% is required. It is possible to detect if any tags has not been read, and (Potdar, Hayati et al. 2007) suggests to use the total weight of the goods to be read, and to compare this with backend information about the weight of the goods that has been read. This is a smart approach, as in some circumstances consistently achieving 100% read rates might not be possible. It is possible to detect if some tags have not been read, and to take action, and it might be easier to detect if some tags are not read than to try and achieve 100% read rates. Sometimes the demand of

100% read rates might interfere with the goal of real-time processing, especially with large quantities (Ding, Li et al. 2007).

5.2.3 READER COLLISION PROBLEM

The reader collision problem is when different RFID readers interfere with each other, either by using the same frequency at the same time or by trying to read the same tag (Engels and Sarma 2002). This is not an unknown problem for wireless communication, and has e.g. been studied extensively for mobile phones. The difference here is that RFID tags are much simpler than mobile phones, especially passive tags, and they can therefore not assist much in the communication.

Solutions to this problem are e.g. to let each reader have its own frequency, or its own timeslot to send a signal. It can also be useful to tune the signal strength of the readers to have minimal overlap. Some overlap is however required to gain full coverage over an area. At a warehouse with many gates with RFID readers, it might be possible to isolate each gate so that radio waves are not transmitted between the gates. The only interference is from other readers at the same gate and not from all the gates (Engels and Sarma 2002).

5.2.4 COST

The price of the tag is another important challenge. This has been a major reason for slow adoption of RFID in the retail industry, as the tag price has been too large compared to the price of the item tagged. The prices have dropped, however, and they will continue to drop as the use of RFID continues to spread. For UHF tags, a major price drop came when the EPC standard was released. It has also helped that large chains such as Wal-Mart have required many of its top suppliers to use EPC for pallet and carton tracking (Liard 2004).

Many people have talked about the “5 cent tag”, and it has been sort of a goal to achieve this price on a tag. This price was first introduced by Sanjay Sarma in 2004 (Sarma 2004).To our knowledge, SmartCode Corp. was the first to achieve this price for EPC Gen 2 tags, at volumes of 100 million tags and above (SmartCode Corp. 2006). This is not very sophisticated tags, and for active tags or passive tags with, e.g., cryptographic features, this price is not yet realistic (Nahas and Deogun 2007).

5.2.5 SECURITY AND PRIVACY

A very important aspect of RFID technology is privacy. There is a great potential for misuse of information when every object is tagged with an RFID. This is mostly a problem when tagging consumer goods, and not when RFID is used for tagging pallets and containers in the supply chain.

Even though the scenario where a thief stands outside a house with an RFID reader and reads information on every object in the house is not realistic, there are other problematic scenarios. While an RFID reader at a store exit can prevent theft of tagged items, it also enables the store to scan each and every customer entering and exiting and to get information on the objects they carry with them. This is not information people want to give away, especially not automatically and without consent. There exist technical solutions for protecting privacy when using RFID, but availability does not mean that they will be used. This is why it is necessary to have rules or laws governing the use of RFID in public areas.

Privacy can also be a problem when used in the supply chain. Consider e.g. trying to keep a shipment secret if all items or pallets shipped carry RFID tags that can be read by anyone. This is not a privacy issue but a security issue.

The European Union has recently released a draft of recommendations for privacy, data protection, and information security principles for RFID supported applications (Rossen 2008). They made the draft available to the public for comments and suggestions, but the consultation is now closed and the draft has been withdrawn in wait for a new version. One of the suggestions is that anyone who wants to implement an RFID application first needs to investigate possible implications on privacy, and to make the results of this publicly available. They also need to publish an easy to understand statement on how RFID will be used. For RFID used in public places, it will be mandatory to inform everyone at the site about this, e.g. by using signs. There will be separate rules for RFID in the retail industry, and it will e.g. be mandatory to deactivate all RFID tags from items sold unless the customer explicitly says otherwise, and it must be possible for the customer to check if this is in fact done (Rossen 2008).

KILL BIT

One technology to improve privacy is a kill bit on the tag. This is a writeable bit in the tag memory that says if the tag is operational or not. If the kill bit is set, no information can be read. The information will still be there but it will not be available to readers. One can for example imagine a situation where it is required to set the kill bit when an item is sold to an end user, which as mentioned above is proposed by the European Union. One problem is that stores might not be careful enough to set the bit on all tags, and that it is difficult for the user to know if it has been done without buying a reader for himself. The end users also looses the possible benefits of the tags, such as washing machines automatically knowing what clothes are being washed and setting the appropriate temperature. Stores might also want the tags active upon a possible return of the item, to see the item’s history (Want 2006) (EPCglobal 2007b).

BLOCKER TAG

The blocker tag is another way to improve privacy. (Ari, Ronald et al. 2003) propose to use a tag that utilizes the singulation algorithm used by the readers to block access to all tags. The most common singulation method is tree-walking, and this is the one used by EPCglobal. In short, this algorithm functions like a depth-first search of a binary tree. It is assumed that the tag identification numbers have a fixed bit length, and that each possible identification number is a node in the tree. The blocking tag blocks this algorithm by simulating all possible tag identification numbers, giving the impression that all tags are present. It is possible to divide the tree into zones, and have the blocking tag block only a sub tree corresponding to a particular zone. This makes it possible to have both public and private tags.

The blocking tag can also be misused by disrupting legit tag reads. A very simple reader will try each possible combination in the tree, and if all tags are present, or seem to be present, the entire tree will be traversed. This will take a very long time, and will result in a denial-of-service attack. For more information on the blocker tag and more advanced uses, see (Ari, Ronald et al. 2003).

5.3 STANDARDS

Different standards exist for how RFID tags are to be constructed, and for communication between the tag and the tag reader. Some of these standards are compatible with each other, and an effort has been made to make different standards more compatible. As an example, EPCglobal has support for various ISO standards in their Tag Air interface. This section presents some important standards for RFID technology.

One of the most used RFID standards is ISO 18000. This is a multipart standard which specifies air interfaces for many frequencies in the LF, HF, and UHF bands. ISO (International Organization for Standardization) also has two other relevant standards, ISO 14443 and ISO 15693. ISO 14443 is a standard for proximity devices with a very short operating range, typically 10 cm, while ISO 15693 is a newer standard in the HF band for vicinity devices with ranges up to 1 meter. ISO 18000-3 is based on this standard (Juels 2006) (Weinstein 2005).

The other major standard comes from EPCglobal. Contrary to ISO 18000, which only specifies air interfaces, EPCglobal has standards for air interface, tag data, middleware and databases for storing and sharing this data and related information. EPC currently only support UHF, but as mentioned above, an air interface using the HF band is under way. The most recent version of EPC UHF Gen 2 has been taken in as a part of ISO 18000-6C, which means that they are compatible and readers are capable of reading both standards. One major requirement for including EPC standards in ISO 18000 was the inclusion of a toggle bit in the tag data standard, which specifies if the tag carries an EPC or if it carries an ISO 15961 Application Family Identifier (O'Connor 2005). As of version 1.3 of the Tag Data Standard, this is included (EPCglobal 2007d). This could mean very much for the adoption of EPC Gen2 tags, as businesses are now guaranteed interoperability with other ISO 18000 standards (Weinstein 2005) (O'Connor 2006a).

The last standard mentioned is NFC from the Near-Field Consortium (NFCIP-1/ECMA340, ISO 18092). As mentioned above, this standard tries to enable mobile or portable devices to communicate with each other. NFC has a tag-reader model, meaning that a device such as a mobile phone is both a tag and a reader. NFC uses the HF band and is compatible with both ISO 14443 and ISO 15693 (Juels 2006).

6 SENSORS

This chapter gives a brief introduction to wireless sensor networks (WSN), applications and challenges of WSNs, and different sensor types. A sensor is a very wide term, and there are many types of sensors. It is possible to consider an RFID reader as a sensor, sensing RFID tags. In this thesis the term sensor will not be used for RFID readers, if not stated explicitly. The sensors presented in this chapter will mainly be related to RFID tags, and the examples are all RFID tags which include sensors on the tag.

Given a fixed computational capacity, the size will in time shrink dramatically. The first computer had a size equal to a house, and today there are computers the size of a credit card or even smaller. It is not hard to believe that there one day will be small sensors everywhere, with acceptable computational capacity to sense the real world; Smart dust and brilliant rocks as M. Satyanarayanan explained it (Satyanarayanan 2003). A network of several inexpensive sensors spread like sand or dust, smart dust, solving small sensor tasks. The other approach is smarter and larger sensors, described as brilliant rocks. Used for calculation and communication between sensors automatically, without the need of an external computer. The technology of today enables the beginning of this revolution of tracking and sensing the world. Not only the use of RFID to track goods in a supply chain, but sensing the context around the goods.

6.1 WIRELESS SENSOR NETWORKS

A wireless sensor network (WSN) can be explained as several sensors in a relation to each other with one or more listeners. The devices in a WSN are constrained with regards to resources and are low on power, processor capacity, bandwidth, storage etc. If the resources of all the devices in a large WSN were combined, it could be a considerable amount. The main aspect of a sensor in a WSN is the ability to:

- Sense data from the physical world
- Record sensor readings, keeping them for later or immediate use
- Process sensor readings to save space and communication need
- Communicate readings to a listener over a wireless network

The third aspect, process senor readings to save space and communication need, is very important when the WSN becomes large. As much processing as possible should be done at the edge of the network, reducing the need for communication. If every possible data sensed had to be sent to a central server for processing, it could be a huge problem with performance and scalability. Ideally, only information wanted and relevant for the rest of the network should be communicated from the nodes. The sensors can either be aware of each other, using themselves and others to communicate their readings to one or more listeners, or they could be unaware of each other communicating directly with the listeners (Culler, Estrin et al. 2004).

6.1.1 APPLICATIONS

Sensors and wireless communication devices become cheaper and smaller. When cost decreases it opens up for a number of uses, enabling widespread use of WSN. These networks can be used to solve several tasks, and (Culler, Estrin et al. 2004) summarizes these in three areas:

- Monitoring space
- Monitoring objects
- Monitoring the interactions of objects with each other and the encompassing space

The following sections will briefly present examples of these three areas.

MONITORING SPACE

Using sensors to monitor and give relevant information about a space. Typical examples are:

- Temperature readings for a cooler or indoor climate control
- Camera surveillance
- Intelligent alarm systems

Temperature sensing is probably one of the most used sensor technologies, there are temperature sensors everywhere. This is used in many different situations, e.g. on processors in computers to know when to turn on the fan and for weather stations monitoring the air temperature.

MONITORING OBJECTS

Using sensors to give relevant information about the given state or condition of an object. Typical examples are:

- Structural monitoring
- Equipment maintenance
- Medical diagnostic

One example of monitoring objects is sensors attached to various types of machines in a factory. These sensors can perform a wide range of measurements, e.g. vibrations or pressure. The sampling rate needs to be much higher for this type of sensing than for environmental sensors. A temperature sensor could for example measure the temperature every minute, but a vibration sensor would have to take measurements much more often.

MONITORING THE INTERACTIONS OF OBJECTS

Using sensors to give relevant information about objects and how they interact with each other. Typical examples are:

- Wildlife habitats
- Disaster management
- Emergency response
- Ubiquitous computing environments
- Asset tracking
- Healthcare
- Manufacturing process flow

This is the most advanced form of sensors, and also the most interesting. It is not enough to monitor an object isolated, but also how that object interacts with other objects and the space it is in. For wildlife habitats it is e.g. a difference between monitoring one cow and an entire herd of cattle, including the interaction between them. This type of sensor use will not be discussed further in this thesis, as it is not relevant. Relevant sensors for this thesis are sensors monitoring space, e.g. the temperature in a cold storage, or sensors monitoring objects, e.g. acceleration sensors.

6.1.2 CHALLENGES

There are several challenges to wireless communication alone, and adding small sensors will drastically increase the complexity. There exist numerous hardware challenges, but this section will focus on the limitations regarding how and for what sensors and WSNs can be used, and how these limitations can be overcome. For more information, see e.g. (Culler, Estrin et al. 2004).

POWER

A major limitation of sensors is power. The power source would occupy most of the sensor’s size and is therefore a challenge. A sensor with a car battery cannot be called small, and a sensor without power would not be able to communicate or sense the world around it. Balancing these factors, size and amount of power, is challenging because of the expected lifetime of a sensor. Sensors would be made to last long, and the minimization of power usage is important. This could be done by disabling functionality in the sensor, such as communication, when it is not used. The usage of the sensor should therefore be taken into consideration, as the use of the sensor defines the power needed.

COMMUNICATION

Due to the complexity of wireless communication, such as packet loss, interference, and distance, power is critical. Transmitting over longer distances or in though environments would drain the battery drastically compared to communication in a perfect setting. As communication consumes most of the sensor's battery power, it should be carefully planned. All sensor readings do not need to be sent the second they are read, and these data can be stored for later transmission. If they are to be stored, the storage capacity of the sensor needs to be considered when planning the use of a sensor. It is necessary to make sure that nothing will be lost. It is also important to send as little irrelevant data as possible. If the receiver discards the data, it should not have been sent. This is a general principle for WSNs; try to make as many decisions as possible at the edge of the network. This is due to scalability and performance, but in this case it also applies to sensor communications. There is also, as mentioned above, a problem with interference, and having sensors only communicating when necessary would reduce the problem with interference from other sensors trying to transmit.

SCALABILITY

When creating a WSN, there will be limitations due to the large amount of sensors. Consider a trailer with pallets stacked with several boxes containing sales items. The numbers start to increase, and it is necessary to plan the communication. The use of super nodes, concatenating sensor readings from several sensors, could be used in a hierarchy to reduce the load. This is also discussed above with regards to making decisions at the sensors instead of centrally.

REMOTE MANAGING

Not all sensors in a WSN would be easy to reach physically, thus remote access would be the only solution if the sensors needed to be reconfigured. Remote access would also decrease the time needed to configure a large number of sensors, because a one-to-one configuration could be dropped. Instead it would be possible to configure a large number of sensors all at once, and as the usage of WSNs increases, the number of sensors will increase drastically, increasing the need for remote management. It will therefore be necessary to find common standards for configuration and management of sensors, to ensure that sensors from different vendors can coexist in a WSN and be managed through the same interface.

6.2 DIFFERENT TYPES OF ELECTRONIC SENSORS

This section gives an overview of some sensors produced by a few manufacturers today. The section will not go into details about how these sensors work; only what they are able to observe to give an idea about what is available. The section will focus on sensors relevant for this thesis, e.g. sensors which can be used to provide context information related to item tracking in general and fresh food specific.

6.2.1 MANUFACTURERS

The sensors described in this section come from the manufacturers CAEN, KSW, and Montalbano. CAEN is an Italian company producing, among other things, RFID readers and tags. They also produce an RFID tag which includes a temperature sensor, called a Temperature Logger UHF Semi-Passive tag (CAEN 2008). KSW focuses on RFID, and delivers both active and passive RFID equipment. They have also included a temperature sensor on their active tag, called VarioSens (KSW 2008). Montalbano produces mainly semi-passive RFID tags which can include many different types of sensors (Montalbano 2008). Montalbano has developed a platform which is modular and makes it possible to include a wide range of sensor types on the RFID tags. Two of them are shown in this section, but they have many other possibilities as well, e.g. pressure and shock sensors.

These manufacturers are selected as examples, and there are many others. It is obvious that the market for RFID tags including environment sensors is growing, as many companies are developing such products.

6.2.2 PRODUCT TYPES

This section presents different types of sensors, with examples from the manufacturers presented above. A set of characteristics is given for each sensor to give a brief description of the performance and qualities of such sensors.

TEMPERATURE

The first, and most used and discussed today, is the temperature sensor. This is very relevant for the fresh food industry, and in many other situations. Each of the three manufacturers produces temperature sensors, and one example from each is included. Each example below is a semi-passive RFID tag including a temperature sensor.

Table 1 Temperature sensors

	CAEN RFID Mod. A927 from CAEN • Time interval for readings is 6 seconds, can be configured on request • Semi passive tag read/write • 3 year lifetime • Supporting ISO 18000, but extended to support sensor data. • Configurable after production • Unix time, internal clock • Two sensors. One of them a probe, enabling internal and external sensing of a box for example.
	VarioSens from KSW • 0 up to 720 values (10Bits per value) • Time interval: 2s up to 9h • Monitoring delay: 0 up to 720 days • Read distance: 0 up to 25cm • 0,6 s for reading the whole data (720 values) • Battery MnO2Zn - printed battery (1,5V ... 1,1V)
	MTsens from Montalbano MTsens measures time and acquires data of exposure to heat with programmable frequency and performs a very accurate data logging.

ACCELERATION

The second example is an acceleration sensor. This type of sensor is able to detect the acceleration of an item, typically by attaching an RFID tag with the sensor to the item. This can be used to detect rough treatment of items, e.g. if it has been tossed or fallen down from a height.

Table 2 Acceleration sensors

	MTshock from Montalbano • Full scale +-2g/6g • -20/+60 degree Celsius • Reusable and low cost • Programmable rate for data acquisition and storage

HUMIDITY

The last example is a humidity sensor sensing the humidity, which is very important to preserve the quality of certain products. The example shown below is from Montalbano, as was the acceleration sensor.

Table 3 Humidity sensors

	MThumidity from Montalbano • Combined temperature and RH% sensing. • 0-100 RH% • Programmable rate for data acquisition and storage • Reporting over 1000 temperature and RH pairs

6.3 CHEMICAL SENSORS

Electronics can be expensive to produce and may therefore be too expensive to be used on every product. A cheaper solution might be the use of chemical sensors, reacting to time and environment. TimeTemp AS is a Norwegian company specializing in making chemical sensors for fresh food. These sensors give the products dynamic expiration dates, which change over time. The TimeTemp tag is shown in Figure 2. It has a dynamic scale going from +14 days to expiration to expired. The expiration date on the tag varies as a function of time and the temperature of the product. This type of tag is not relevant for use with RFID tags and sensor networks, as it has no means to communicate its observations, but is included to show an alternative approach.

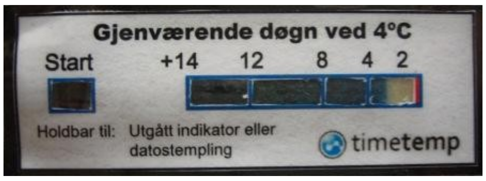


Figure 2 Chemical sensor (TimeTemp 2008)

7 INTEGRATING RFID AND SENSOR SYSTEMS WITH EPC/EPCIS

This chapter will describe how RFID systems can be integrated with the EPC standards from EPCglobal. It will not go into detail on what EPC and EPCIS is, but a chapter from a previous report (Albretsen and Larsen 2007) is included in Appendix B. Some of the extension mechanisms in EPC that can be used to include sensor data as well as other information in addition to the RFID already there will be discussed. It will also be explained how systems can utilize EPCIS to share this data. The use of EPCIS is independent of the underlying tag standard used. EPCIS can be used both with EPC tags and other tags. Figure 3 shows an overview of the EPCglobal Architecture Framework.

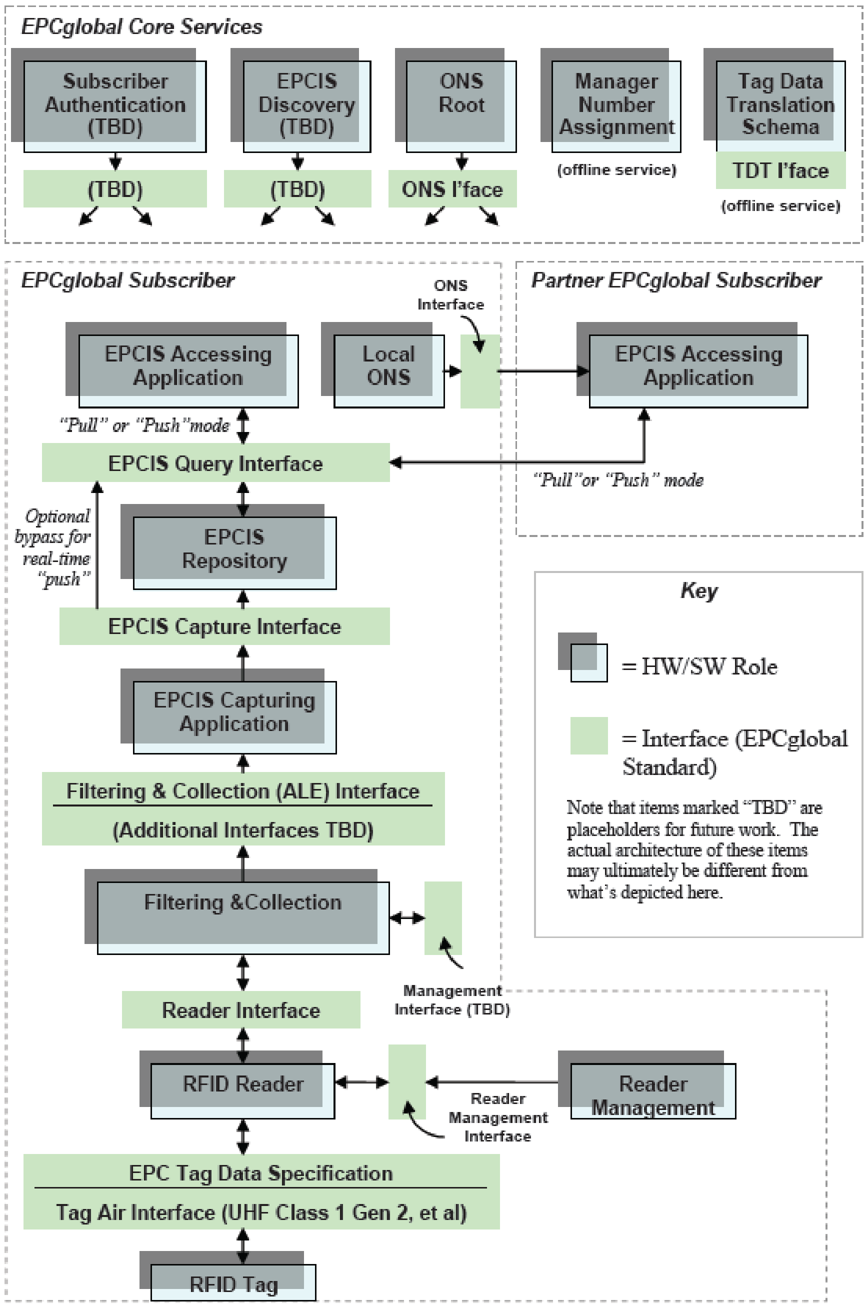

EPCglobal Core Services
Subscriber Authentication (TBD)
(TBD)
EPCIS Discovery (TBD)
(TBD)
ONS Root
ONS I'face
Manager Number Assignment
(offline service)
Tag Data Translation Schema
TDT I'face
(offline service)
EPCglobal Subscriber
ONS Interface
EPCIS Accessing Application
Local ONS
"Pull" or "Push" mode
EPCIS Query Interface
Optional bypass for real-time "push"
EPCIS Repository
EPCIS Capture Interface
EPCIS Capturing Application
Filtering & Collection (ALE) Interface
(Additional Interfaces TBD)
Filtering &Collection
Management Interface (TBD)
Reader Interface
RFID Reader
Reader Management
Reader Management Interface
EPC Tag Data Specification
Tag Air Interface (UHF Class 1 Gen 2, et al)
RFID Tag
Partner EPCglobal Subscriber
EPCIS Accessing Application
"Pull" or "Push" mode
Key
= HW/SW Role
= Interface (EPCglobal Standard)
Note that items marked "TBD" are placeholders for future work. The actual architecture of these items may ultimately be different from what's depicted here.

Figure 3 EPCglobal Architecture Framework (EPCglobal 2007c)

7.1 INCLUDING USER DATA WITH STANDARD EPC TAGS

This section will explain how it possible to include sensor data in addition to the RFID on the tag using the EPC tag standard. The Class 1 Gen 2 Tag Air Interface specifies four different memory types, or memory banks, which are: reserved memory, EPC memory, TID memory and user memory. The first three memory types are predefined by the standard, but the user memory is available for extra information. For EPCglobal applications, this memory must be encoded according to the EPC Tag Data Standard, for other applications it must conform to ISO/IEC 15961 and ISO/IEC 15962. The current version of the Tag Data Standard (version 1.3.1) does not specify how EPCglobal applications should encode the user memory, but it is expected to be addressed in a future version (EPCglobal 2007d) (EPCglobal 2007b).

The user memory makes it possible to include industry specific data on the tag, and as such makes it unnecessary to have one tag for the EPC and one for an industry specific code. An example of use of this is within the car industry in the U.S. The industry plans to include the U.S. Department of Transportation tire identification number, with information on production date and location, in the user memory. This is just one example, and other industries can include different information in the same memory. As the EPC standard specifies that the user memory must conform to the ISO 15961 and 15962 protocols, RFID readers will know how to access the user data even though it does not know what kind of data is included. This is important as reader manufacturers only need to conform to one standard (in addition to EPC). Different industries can include the information they want as long as it is encoded according to the same standard (O'Connor 2006b) (Harmon 2006).

Other possible uses of the user memory are sensor data, expiration dates, serial numbers etc. All data relevant for a unique item and not the entire class of items is potential data for the user memory, limited by the size of the memory. A problem with large user memories can be the transmit time of the data, which will have an impact on tag throughput of a system.

The EPC standard does not specify the size of the tag memory, other than that each of the four memory banks shall contain zero or more 16 bit memory words, or how the memory is organized. These are issues that should be addressed to achieve a good utilization of the user memory. An EPC for identification is very useful, but more information is also necessary to get higher benefits from RFID tags. It is therefore important that the specifications for the user memory are more detailed, to make it easier to use (Harmon 2006).

7.2 INTEGRATING WITH EPCIS

The other part of the EPCglobal standards discussed here is EPCIS, used to publish and retrieve information related to EPCs (EPCglobal 2007a). If EPCIS gets a wide adoption, it will be interesting for companies and industries using any tag standard to integrate with this architecture, both for EPC and non-EPC tags. If, e.g., a retail chain shares and retrieves product information using EPCIS, it would want to use this from all its suppliers, independent of the suppliers' underlying systems. This section will first explain how the underlying tag type used is not relevant for the use of EPCIS, and then look at how the EPCIS standard can be adapted to suit the needs of different industries. Figure 4 shows the different layers in EPCIS.

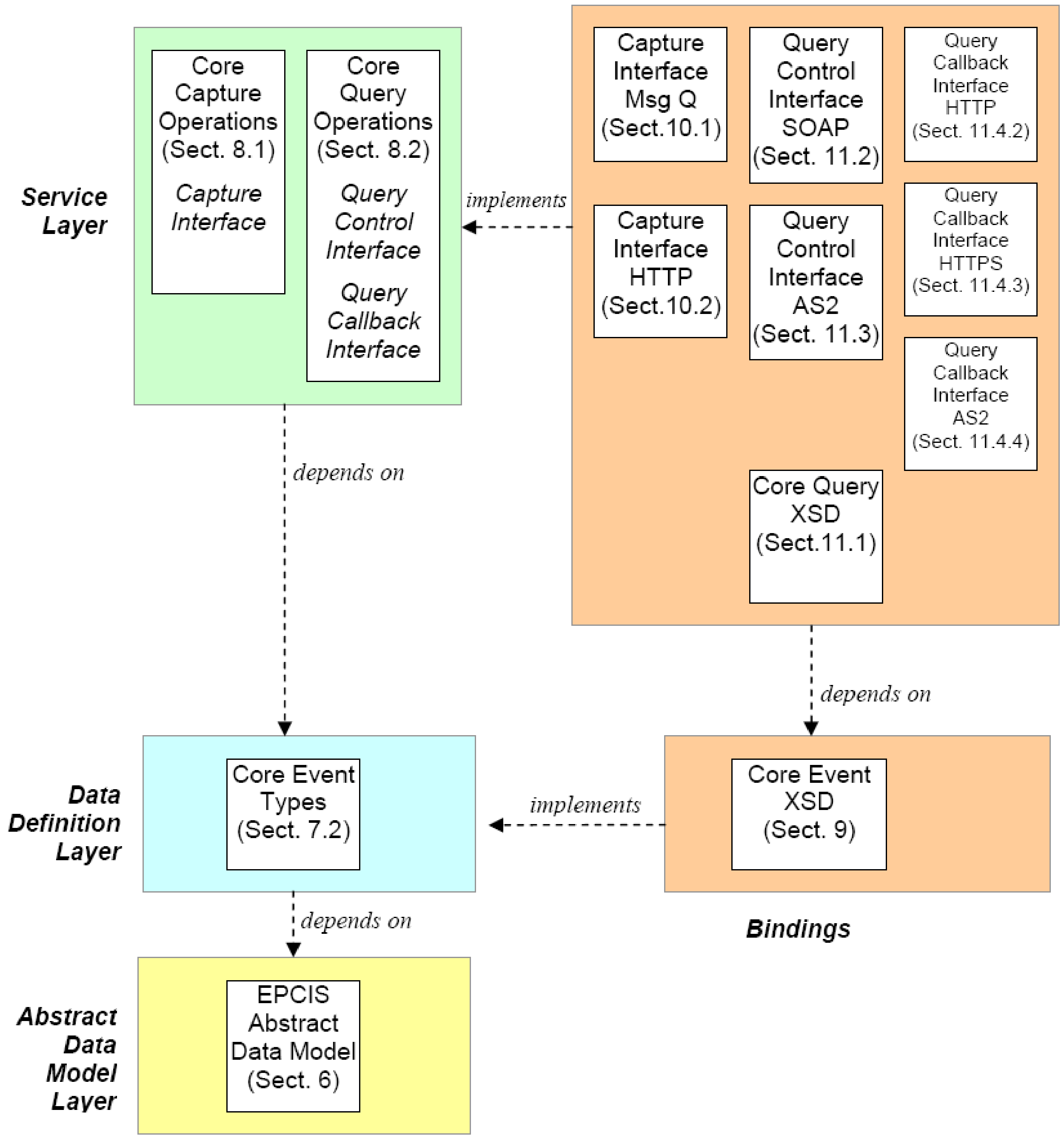

Service Layer
Core Capture Operations (Sect. 8.1)
Capture Interface
Core Query Operations (Sect. 8.2)
Query Control Interface
Query Callback Interface
implements
Capture Interface Msg Q (Sect.10.1)
Query Control Interface SOAP (Sect. 11.2)
Query Callback Interface HTTP (Sect. 11.4.2)
Capture Interface HTTP (Sect.10.2)
Query Control Interface AS2 (Sect. 11.3)
Query Callback Interface HTTPS (Sect. 11.4.3)
Query Callback Interface AS2 (Sect. 11.4.4)
Core Query XSD (Sect.11.1)
depends on
depends on
Data Definition Layer
Core Event Types (Sect. 7.2)
implements
Core Event XSD (Sect. 9)
depends on
Bindings
Abstract Data Model Layer
EPCIS Abstract Data Model (Sect. 6)

Figure 4 EPCIS layers (EPCglobal 2007a)

7.2.1 USING EPCIS INDEPENDENT OF TAG TYPE

The first EPCIS part of the EPC Architecture Framework is the EPCIS Capturing Application, as seen in Figure 3. Everything below the Capturing Application is related to RFID tags, the reading of tags, and filtering and collection of tag reads. The reason that EPCIS is independent of tag type is that it is not specified explicitly how an EPCIS Capturing Application retrieves its information. The EPCglobal Architecture standard specifies the responsibilities of that role (among others) as:

> *"[EPCIS Capturing Application (Role)] may coordinate multiple sources of data in the course of recognizing an individual EPCIS event. Sources of data may include filtered, collected EPC data obtained through the Filtering & Collection Interface, other device-generated data such as barcode data, human input, and data gathered from other software systems."*(EPCglobal 2007c)

This means that there are no reasons that the data could not come from another source than the EPC Filtering & Collection interface. The EPCIS standard also specifies that an EPCIS Event does not have to directly relate to a physical tag observation. It is therefore no problem to use EPCIS to share information from many different sources, both standard EPC tags and others. The only limitation is that the item specified should have an EPC code. It could be possible to use EPCIS and its extensions with another code standard than EPC, but then it would not be necessary to use EPCIS at all.

One of the benefits of using EPCIS is using the Object Name Service (ONS) and EPCIS discovery to find information about a specific item based on the EPC code. A retail store can e.g. use the same interface to retrieve information on each of its products, as long as the product has an EPC number. It is irrelevant whether the EPC is encoded by an EPC Class 1 Gen 2 UHF tag, another RFID standard tag, a barcode or by other means.

7.2.2 EXTENDING EPCIS TO FIT DIFFERENT NEEDS

The entire EPCIS standard has a wide focus on extensibility. Extensibility is one of three principles defined for the specification of EPCIS (in addition to layered and modular). It should therefore be possible for most industries to adapt it to their needs.

The standard specifies two mechanisms used for extensibility: subclassing and extension points. In addition to this, there is extensibility inherent in the use of a layered approach. By using subclassing, it is possible to extend every data type in the Data Definition Layer and include additional functionality or information in the class, while still being able to work together with users that only know of the parent class. This ensures backward compatibility, as the subclass only need to be known by those wishing to use the extra functionality or information. Such a subclass can be used wherever the parent class is used. Extension points are included for vendors to provide extended functionality without using subclasses. The standard clearly specifies where extension points are required or optional. Wherever an extension point occurs, implementations shall provide for extensibility through new operations. Extensions from EPCglobal in future versions of the standard and from proprietary vendors shall be supported. Extensibility from layering comes from the fact that each layer may be implemented in many different ways, but is still able to interoperate. The obvious example from EPCIS is the different data bindings provided. If one industry wants to use another binding, it is no problem to implement a new binding type and reuse the rest of the layers.

The Abstract Data Model layer defines the two data types “event data” and “master data”. The standard specifies the structure of these data types, but not the meaning. Event data can be seen as data that grows over time as business is conducted, and it is on the form “Item A passed location X at time T”. Master data is more static and is not expected to be updated very often. Master data is there to help interpret the event data, and in the case above, could explain what item A is, and where location X is.

These two data types lay the foundation for all data processing in EPCIS, and there are 5 methods defined for extending the data processed. These methods allow the creation of:

- New event types
- New event fields
- New vocabulary types
- New master data attributes
- New vocabulary elements

An event type is defined as nothing more than a namespace-qualified name (qname) that indicates which event type a given event conforms to. The event types are defined in the Data Definition Layer. An event field is a qname that refers to a field in an event type defined in the Data Definition Layer, or to an extension. A vocabulary is a set of alternative values for an event field, and vocabulary elements are items in such a vocabulary. An example of a vocabulary can be a set of available location names. Master data attributes are name/value pairs that are associated with a vocabulary element. See Table 4 for an overview of the extension mechanisms. The table also specifies how these extensions are made available to the users.

Table 4 EPCIS data extension methods (EPCglobal 2007a)

Organization type	Extension Method					How Disseminated
	New Event Type	New Event Field	New Vocabulary Type	New Master Data Attribute	New Vocabulary Element	
EPCglobal EPCIS Working Group	Yes	Yes	Yes	Occasionally	Rarely	New Version of EPCIS Spec
EPCglobal Business Action Group for a specific industry	Rarely	Rarely	Occasionally	Yes	Yes (standard vocabulary)	Specification Document
Industry Consortium or Private End User Group outside EPCglobal	Rarely	Rarely	Occasionally	Yes	Yes (standard vocabulary)	Private Group Interoperability Specification
Individual End User	Rarely	Rarely	Rarely	Rarely	Yes (user vocabulary)	Updated Master Data via EPCIS Query or other data sync

There are many possibilities for extensions, and it is clearly defined which organization should define an extension to each part of the standard. It is worth noticing that individual users do not have many options for extending the specification. This is because an extension is not worth anything if only one part is aware of its existence, thus such extensions should be coordinated by e.g. an industry consortium. For large corporations this might be different, as they can control all parts of the value chain, and they will sometimes not need to exchange data with other corporations. Such corporations might benefit from individual extensions, but in most cases this should be coordinated by a larger group.

Throughout the standard, it is clear that EPCglobal tries to accommodate as many industries as possible, by making the standard very generic. It is also possible to create non standard extensions to meet specific industry needs that might be used to expand and update the standard specification in a later version (EPCglobal 2007a).

7.2.3 EPCIS INTERFACES

The EPCIS standard (EPCglobal 2007a) specifies three interfaces and one data model. The data model is a standard way to represent information on physical objects, described above. To integrate with EPCIS it is also necessary to implement the three interfaces, one for capturing event data and two to retrieve this information. Each of these interfaces will be explained below.

7.2.3.1 EPCIS CAPTURE INTERFACE

The capture interface is used by EPCIS Capture applications to provide EPCIS Events to a repository or other applications. This is a very simple interface with only one method which takes a list of EPCIS events as an argument.

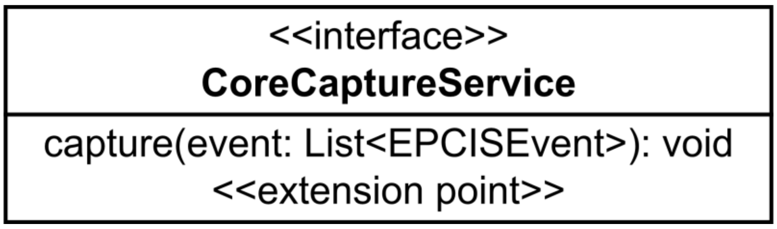
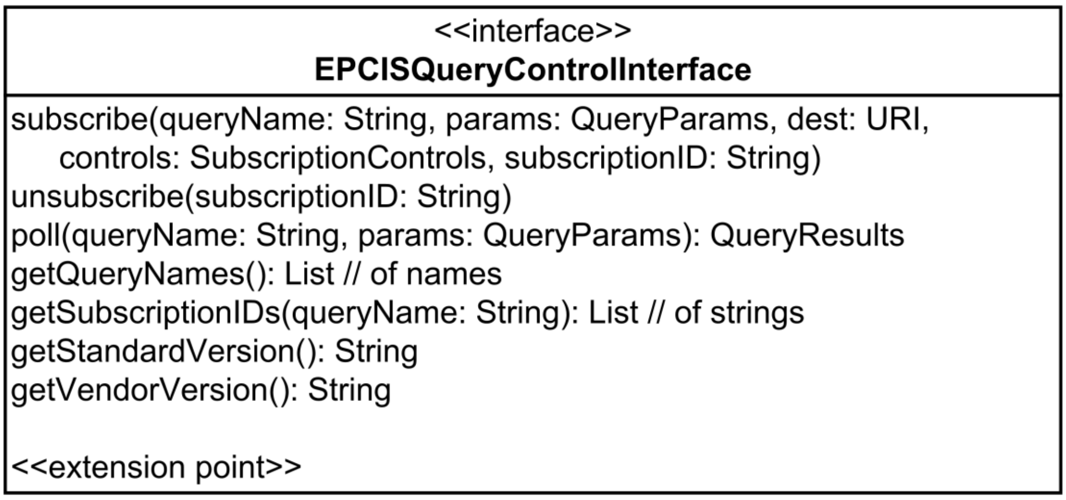


Figure 5 EPCIS Capture Interface (EPCglobal 2007a)

The interface has the extension point mentioned above, and it will also accept subtypes of the EPCISEvent class, which means that vendor specific events will be accepted through the standard interface. This is the entire capture module of the standard. The EPCIS Capture Application from the EPC AF (Figure 3) is a role and not an interface, and is therefore not specified. This is where RFID data is retrieved through the Application Level Events (ALE) interface or by other means, and where the EPCIS Events are generated. How this is achieved is up to the implementation, the only requirement is that the events are delivered through this capture interface. It is not specified how master data enters the system; it is assumed static in comparison and needs to be entered by other means.

7.2.3.2 EPCIS QUERY INTERFACES

The query operations module consists of two interfaces, the *EPCIS Query Control Interface* and the *EPCIS Query Callback Interface*. These are used by EPCIS Accessing applications to retrieve EPCIS data. The Query Control interface is used to retrieve EPCIS data from a repository, and to subscribe to standing queries to receive push information. This is the only interface (of these two) that needs to be implemented by the service itself. The Query Callback interface is the means through which the service delivers results from standing queries subscribed to by accessing applications.



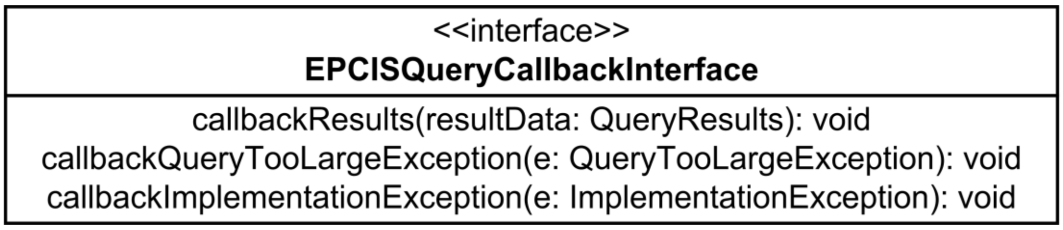
```
<<interface>>
EPCISQueryControlInterface

subscribe(queryName: String, params: QueryParams, dest: URI,
    controls: SubscriptionControls, subscriptionID: String)
unsubscribe(subscriptionID: String)
poll(queryName: String, params: QueryParams): QueryResults
getQueryNames(): List // of names
getSubscriptionIDs(queryName: String): List // of strings
getStandardVersion(): String
getVendorVersion(): String

<<extension point>>
```

Figure 6 EPCIS Query Control Interface

The *EPCIS Query Control Interface* consists of seven methods. Four of these only return lists of names or version IDs. The version methods are used by applications to find which version is supported by the service, and backwards compatibility is assumed. The other three methods are more interesting: *subscribe*, *unsubscribe* and *poll*. *Poll* is the method used to retrieve data directly from a repository. It has only two parameters; the name of the query and the parameters for that query. This is because EPCIS only supports predefined queries. The standard specifies two, *SimpleEventQuery* and *SimpleMasterDataQuery*, but it is possible for vendors to add others. The *QueryParams* parameter is simply name/value pairs corresponding to the query specified by the query name. The method returns a *QueryResults* object, which consists of the query name, a subscription ID and a *QueryResultsBody* specified independently for each query. The subscription ID shall actually be omitted for the *poll* method, but is included in results for standing queries. *Subscribe* is used to register for a standing query to receive "push" information. All the parameters from *poll* are included, and in addition *dest* which specifies where to deliver the results, *SubscriptionControls* which defines the schedule for the query and *subscriptionID*. *Unsubscribe* simply removes a previously registered subscription.



```
<<interface>>
EPCISQueryCallbackInterface

callbackResults(resultData: QueryResults): void
callbackQueryTooLargeException(e: QueryTooLargeException): void
callbackImplementationException(e: ImplementationException): void
```

Figure 7 EPCIS Query Callback Interface

The query callback interface is invoked by services to deliver results of standing queries to accessing applications. *CallbackResults* is used to deliver the query results, with the same *QueryResults* object as described above. As this is standing queries, and not replies to a direct request, exceptions needs to be handled in a special way. In the query callback interface this is done by implementing two methods which is invoked instead of the *callbackResults* method when an exception occurs. There are two possible exceptions, *queryTooLargeException* and *implementationException*, and there is one method for each. These are invoked by the service if an exception occurs when processing a standing query.

This concludes the chapter on how RFID and sensor systems can be integrated with EPCIS. The next chapter will present important issues regarding the administration of RFID and sensor networks.

8 ADMINISTRATION OF RFID AND SENSOR NETWORKS

It is obvious that there are many benefits from using RFID and sensors to track and monitor goods throughout the value chain, but there are also challenges. One of these is the administration of large RFID/sensor networks. Depending on the number of items tracked and the sampling rate of the sensors, an enormous amount of data will be generated, and this needs to be managed. There are also a potentially very large number of readers, tags, and sensors that needs to be administered, and this should be done as generic as possible. Even when using equipment from different vendors, it would be a goal to administer and monitor these through a common interface. This is why a middleware system is needed. There is a big difference between a few readers and tags in a pilot implementation to a full scale deployment in a global corporation. This chapter will first look at two different solutions from EdgeWare and Nofilis. They both define a set of challenges for RFID solutions, and describe how they try to solve them. The focus here is on the challenges and requirements for RFID middleware, and not on the specific products they promote.

The different areas of focus from the two articles are combined below (Hanebeck 2006) (Nofilis 2007):

1. **Manage your data volume**
   An enormous amount of data will be generated, and this needs to be managed. It will therefore be necessary to have some kind of business logic at the edge layers, which filters the data going through to the enterprise systems. The amount of data is so large that it will be important to ensure that only relevant data will be sent through the network.
2. **Determine your need for “local” intelligence**
   As mentioned earlier, the reading of an ID in itself is not very interesting; it must be combined with other information. It is at least necessary to know to which item an ID belongs. One might also want to relate the reading of the ID to a purchase order or an invoice, and it might be interesting to know why exactly that item was detected at a given location.
3. **Define your need for configuration flexibility**
   Configuration of the RFID network is important, and users need to consider the different processes that will take place at the different locations. It might also be a demand for different treatment of different customers at the same location. This can e.g. be done using workflow scenarios.
4. **What else do you need? Data attributes!**
   There are a lot of possible data attributes not related to the product itself that might be interesting. Some examples are at which reader an item was last detected, where it was first detected and entered the system, date and time of readings, items’ relative positions to other items etc.

5. **How do you manage historical data?**
   It will be necessary to retrieve information based on individual IDs, and existing enterprise systems might not support this. It can be very expensive to exchange all legacy systems, and it might be cheaper to develop a new system that integrates with the existing one to handle such data.
6. **Assess your need for reverse read loops:**
   This is related to exceptions in business processes. When an exception condition occurs, the process needs to be reversed. Exceptions can, e.g., be damaged goods, difference between read goods and what was supposed to be read, missing goods. Some reversals can be done automatically, others require manual intervention. It is therefore important to have monitoring solutions in place, and support for alerts and notifications to operators.
7. **How do you enable collaborative processes?**
   This relates to using the data provided by RFID in collaborative processes. It is for example possible to collect performance metrics and to share these with other parts of the supply chain to optimize the chain. Another example could be a retail store wanting product information from a manufacturer, and this information is collected and lies in the RFID network of that company. When enabling such collaborations, security becomes a very important consideration.
8. **Health monitoring & remote management**
   Health monitoring needs to be done automatically as manual overview of the entire network does not scale at all. It should be possible to know which sensors break, what is wrong with them, and where they are as soon as possible after a problem occurs. Remote management is especially important when the network is geographically spread, as it is necessary to be able to manage the devices from a central solution. It is also a great benefit to be able to create standardized configurations that can be deployed to many readers, independently of the brand or type of reader.
9. **RFID application development and integration**
   Once the solution is deployed, it must be possible to develop and integrate new applications into the solution without disrupting the daily use. This can for example be achieved by the possibility to emulate the entire physical network to test new applications.
10. **RFID "Plug & Play"**
    RFID Plug & Play means that it should be possible to administer the physical infrastructure and the business processes independently. The addition of new RFID devices should be done without disrupting business processes, and business processes should be changed independently of the infrastructure.

Some of the points above are issues discussed earlier. One issue is the importance of configurability and monitoring of the system. Without this support, it will be impossible to control a large system. The other is the need to share the data acquired. The greatest potential in RFID solutions lies in sharing the information between business partners. With time it might also be possible and interesting to share this with end users in a retail store that want more information about a product they are interested in buying.

It is important to find ways to manage both the RFID infrastructure and the amount of data generated. This must be done in a way that supports heterogeneous devices, collaboration with other corporations, and makes it easy to monitor the performance of the system. It is also important to clearly define what kind of data will be needed, and how they will be used. The amount of data may be very large, and it is necessary to only process and store information that will be of use. It might be difficult to determine what information and data attributes is needed in the future. Without some control over the data, it will be very difficult to manage such systems. For the same reasons, some kind of intelligent layer at the edge is needed. The system needs to determine what kind of information it wants when the RFIDs are read, not after each reading is processed and distributed across the network. If each such decision is to be processed by a central server, the system would not scale.

According to Forrester Research, RFID middleware solutions need to support the following seven capabilities (Leaver 2004):

- Reader and device management
- Data management
- Application integration
- Partner integration
- Process management and application development
- Packaged RFID content
- Architecture scalability and administration

These are mainly the same areas as covered above. The one not discussed previously is "Packaged RFID content". This is a requirement that the middleware is able to integrate with RFID-related applications and understand processes such as shipping and asset tracking. With time, the area of application for RFID middleware will change as readers become more and more intelligent. It is probable, e.g., that more and more filtering will be done at the readers to limit network load. This does not mean that all filtering will be done at the reader, some filtering still needs to be done at a higher level in the system. As more standards for RFID readers evolve, the need for each enterprise to find standardized ways of configuration for its readers might change, as they can follow global standards. A way to centrally manage and administer the readers will still be needed, and this can be done by the middleware (Leaver 2004).

9 SENSOR WEB ENABLEMENT STANDARDS

A sensor network is many different sensors at different locations that are connected and accessible by computers. A Sensor Web refers in this case to sensor networks that are accessible over the web using standard protocols and APIs. The idea is to have a multitude of sensors available online monitoring many different conditions, and to provide metadata describing the live and stored data to discover, access, and use these data from anyone using a web browser. It should also be possible to control these sensors remotely via the web for configuration and maintenance (Reichardt 2003) (Botts, Percivall et al. 2007). Figure 8 shows the concept of a sensor web.

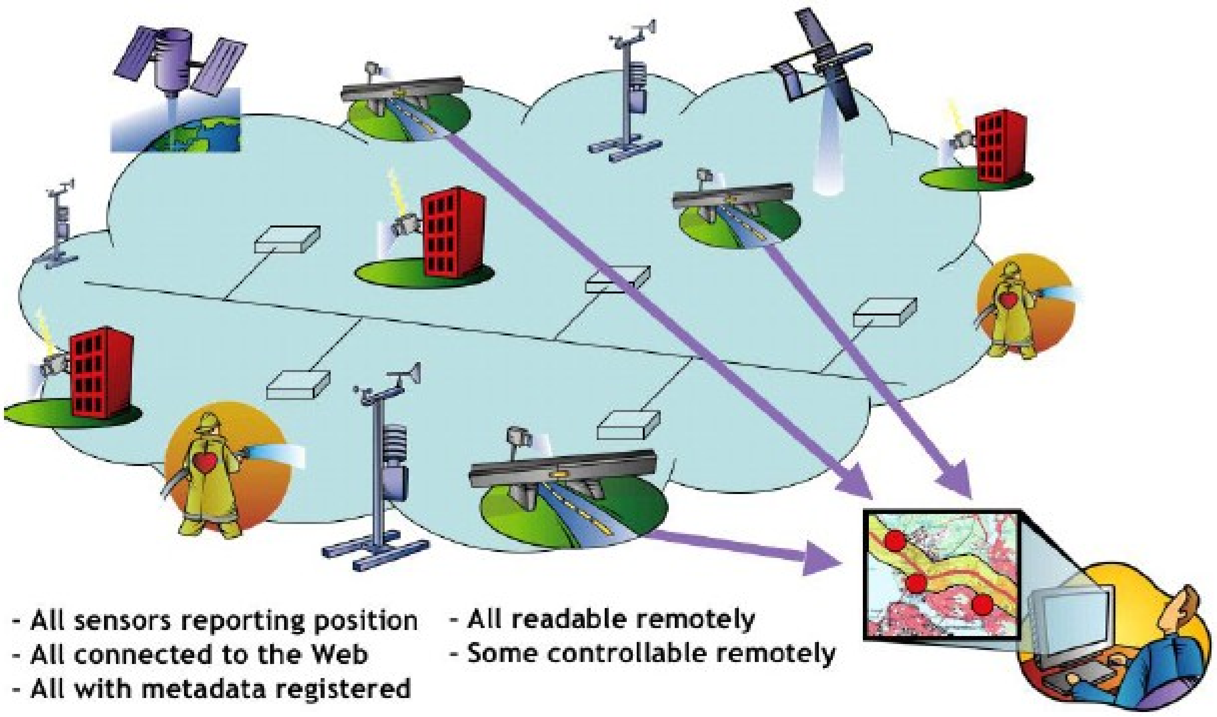

- All sensors reporting position
- All connected to the Web
- All with metadata registered
- All readable remotely
- Some controllable remotely

Figure 8 Sensor Web example (Botts, Percivall et al. 2007)

Open Geospatial Consortium (OGC) is an international standards organization similar to EPCglobal. OGC develops standards for geospatial and location-based services. They have an initiative called “Sensor Web Enablement” (SWE) where they are building a framework of open standards to support the sensor web. There are mainly six areas of functionality targeted by this architecture (Botts, Percivall et al. 2007):

- Discovery of sensor systems, observations, and observation processes that meet an application’s or user’s immediate needs;
- Determination of a sensor’s capabilities and quality of measurements;
- Access to sensor parameters that automatically allow software to process and geo-locate observations;

- Retrieval of real-time or time-series observations and coverage in standard encodings;
- Tasking of sensors to acquire observations of interest;
- Subscription to and publishing of alerts to be issued by sensors or sensor services based upon certain criteria.

Support for this functionality is enabled by a set of specifications for encoding sensors and sensor observations, and several service interfaces using web services. Every standard is based on XML. There are currently seven different standards created by the SWE working group, which all are pending OpenGIS Specifications (Botts, Percivall et al. 2007):

1. **Observations & Measurements Schema (O&M)** – Standard models and XML Schema for encoding observations and measurements from a sensor, both archived and real-time.
2. **Sensor Model Language (SensorML)** – Standard models and XML Schema for describing sensors systems and processes; provides information needed for discovery of sensors, location of sensor observations, processing of low-level sensor observations, and listing of taskable properties.
3. **Transducer Markup Language (TransducerML or TML)** – The conceptual model and XML Schema for describing transducers and supporting real-time streaming of data to and from sensor systems.
4. **Sensor Observations Service (SOS)** - Standard web service interface for requesting, filtering, and retrieving observations and sensor system information. This is the intermediary between a client and an observation repository or near real-time sensor channel.
5. **Sensor Planning Service (SPS)** – Standard web service interface for requesting user-driven acquisitions and observations. This is the intermediary between a client and a sensor collection management environment.
6. **Sensor Alert Service (SAS)** – Standard web service interface for publishing and subscribing to alerts from sensors.
7. **Web Notification Services (WNS)** – Standard web service interface for asynchronous delivery of messages or alerts from SAS and SPS web services and other elements of service workflows.

The first three standards are XML Schemas for encoding of sensors, sensor observations, and real-time streaming to and from sensor systems. These three are required to have a common understanding of the functionality of a sensor, and what the data coming from the sensors mean. The last four standards are web service interfaces enabling communication with the sensors. The parts of the framework most important for this thesis will be described in the following sections. Many of the standards rely on other standards, e.g. various ISO standards, but these will not be explained.

9.1 OBSERVATIONS AND MEASUREMENTS (O&M)

Observations and Measurements is a standard focusing on representation and exchange of observation results. By having a common standard for this it will not be necessary to implement support for a long range of vendor-specific data formats for each sensor in the sensor network. This section will give a brief introduction to O&M, for more details see the standard document (Open Geospatial Consortium 2007a).

Observations and Measurements have at its core a basic observation type. An UML model of this type is shown in Figure 9.

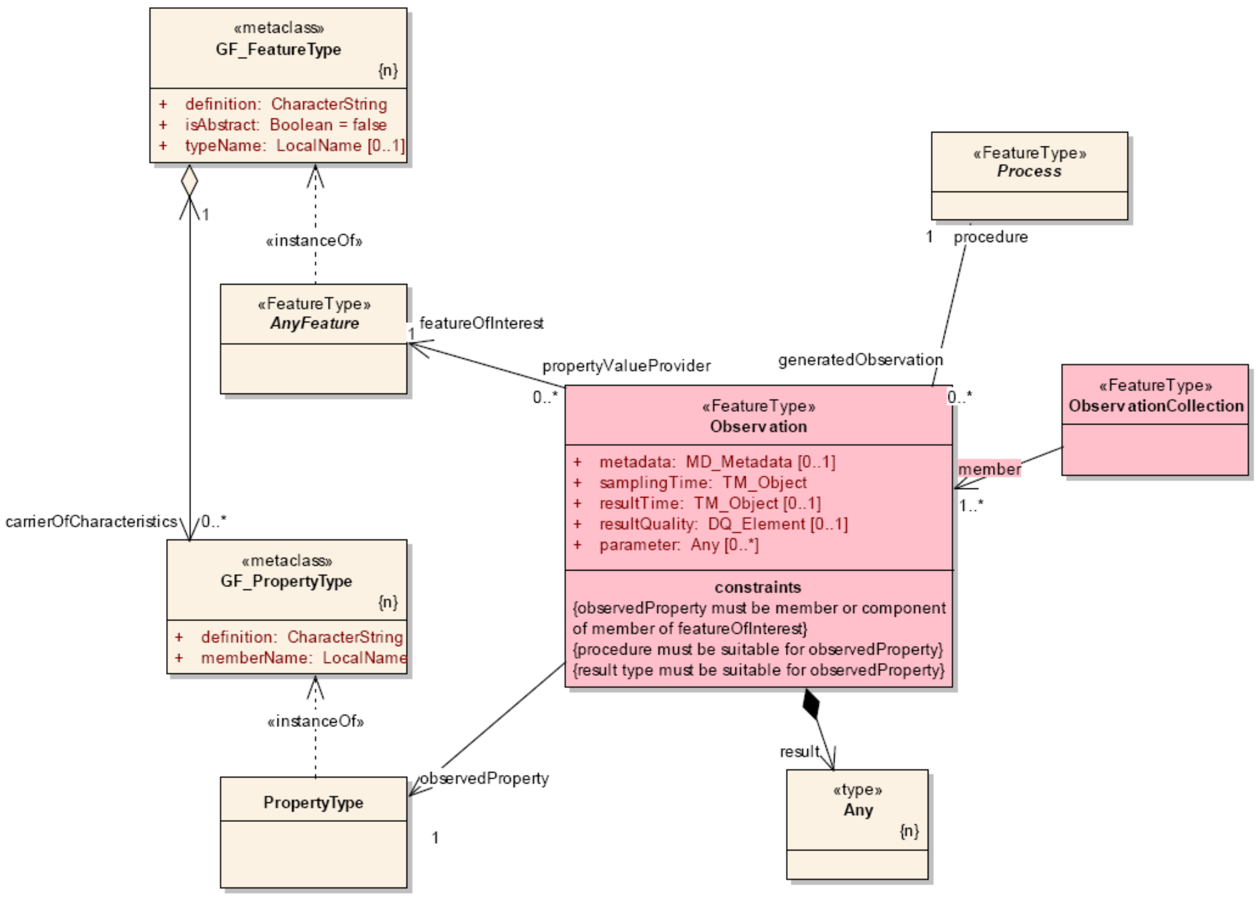

«metaclass»
GF_FeatureType
{n}
+ definition: CharacterString
+ isAbstract: Boolean = false
+ typeName: LocalName [0..1]
1
«instanceOf»
«FeatureType»
Process
1 procedure
«FeatureType»
AnyFeature
1 featureOfInterest
propertyValueProvider
generatedObservation
0..*
0..*
«FeatureType»
Observation
+ metadata: MD_Metadata [0..1]
+ samplingTime: TM_Object
+ resultTime: TM_Object [0..1]
+ resultQuality: DQ_Element [0..1]
+ parameter: Any [0..*]
constraints
{observedProperty must be member or component of member of featureOfInterest}
{procedure must be suitable for observedProperty}
{result type must be suitable for observedProperty}
«FeatureType»
ObservationCollection
member
1..*
carrierOfCharacteristics
0..*
«metaclass»
GF_PropertyType
{n}
+ definition: CharacterString
+ memberName: LocalName
«instanceOf»
PropertyType
observedProperty
1
result
«type»
Any
{n}

Figure 9 Basic Observation type (Open Geospatial Consortium 2007a)

This type has four key properties:

- The featureOfInterest is a representation of the object regarding which the observation is made.
- The observedProperty is the property associated with the feature of interest, and describes the phenomenon observed.
- The procedure is a description of the process used to get the result.

- The result is the value received. The type of the result must be consistent with the property observed.

Table 5 shows an example on these four properties related to earth observations.

Table 5 O&M example

O&M	Earth Observation
Observation->Result	Observation value, measurement value
Observation->Procedure	Method, sensor
Observation->ObservedProperty	Parameter, variable
Observation->FeatureOfInterest	Media (air, water…)

The standard describes the observation as a *"property-value-provider"* for the feature of interest. This means that the result contains the observed value of a given property of the feature. The other detailed information in the observation type is of more interest for applications that evaluate errors in the estimated values. The feature of interest type is further specified in (Open Geospatial Consortium 2007b).

Domain specialization will be needed in any actual implementation. This will primarily be done by the associated classes, and not by the observation class itself. These classes are referred to as the *"second layer"* in the standard. E.g. where the model says <<FeatureType>>, this will be determined by actual feature instances for that application. Detailed schemas for these second-layer classes are generally domain-specific schemas utilizing O&M. For examples on how such schemas can be implemented, see (Open Geospatial Consortium 2007a).

9.2 SENSOR MODEL LANGUAGE (SENSORML)

Sensor Model Language is a framework used to describe sensor systems and the processes associated with these (Open Geospatial Consortium 2007c). The goal is to be able to describe almost any sensor system to treat any sensor in a heterogeneous sensor network in the same way. The concept behind SensorML was first proposed and implemented in SPICE, a software system developed by the Navigation and Ancillary Information Facility at NASA JPL. It has later been further developed at the University of Alabama in Huntsville, before it became an OpenGIS standard.

The information provided by SensorML is in essence the information needed to interpret observation results. If, e.g., a distance sensor measures the distance to something to be 5 meters, it is necessary to know where the sensor is located, the angle the measurement is taken etc. This can be provided by SensorML. Even though SensorML is a part of the SWE framework it can also be used alone or together with other sensor system architectures. It is however not capable of describing observation results. A standard such as O&M, described above, is thus necessary. SensorML is used in the SWE framework to enable sensor discovery, including capabilities, location, and taskability. It also helps processing real-time observation data, both by SensorML-enabled software and intelligent sensors using SensorML, e.g. to determine the location of its observations. Finally, it describes interfaces and taskable parameters to enable tasking services, and it provides sensor information to accompany sensor alerts sent by sensor systems.

SensorML is actually a process modeling language capable of describing any process. This includes the process used for sensor measurements and post-processing. SensorML models everything as processes. Such process models define inputs, outputs, parameters, and methods for that process as well as metadata describing the process. They are all defined using the SWE Common Conceptual Models. The SWE common provides the following categories of data types (Open Geospatial Consortium 2007c):

- Primitive data types, complementing those implemented in the Geography Markup Language (GML)
- General purpose aggregate data types, including records, arrays, vectors and matrices
- Aggregate data types with specialized semantics, including position, curve, and time-aggregates
- Standard encodings to add semantics, quality indication and constraints to both primitive and aggregate types
- Specialized components to support semantic definitions, as required above
- A notation for the description of XML and non-XML array encodings

SWE Common are currently a part of the SensorML standard, but an OpenGIS Work Group is working on extracting it to a separate standard (Botts, Percivall et al. 2007) (OpenGIS 2008).

SENSORS AND DETECTORS

SensorML distinguishes between sensors and detectors. Detectors are devices which take an input signal and generate one or more outputs. Detectors are based on simple models. The simple processes in SensorML are called “process models” or “components”. A process model is used for pure processes, while components are used for physical processing devices, such as detectors.

Sensors are a combination of several detectors linked in parallel or series. As detectors are modeled as processes, sensors can be seen as process chains. In SensorML, the term “process chain” is used for processes that do not exist in the physical domain, and “sensor systems” are used for physical devices. Systems are a collection of sensors that are somehow related spatially and temporally. SensorML utilizes the composite design pattern (Gamma 1994), and both process chains and sensor systems are processes, which mean that they have inputs, outputs, parameters, and methods. This enables collections of process chains and sensor systems. This thesis will mainly focus on sensor systems and not process chains.

The creation of sensor systems, or sensor platforms as the standard also describes it, enables the developer to create metadata concerning the entire system. It is possible to relate the sensors to each other, effectively creating a network of related sensors e.g. to generate temperature profiles for a given space. The typical situation will be that sensors are modeled as containing one or more detectors, and systems contain one or more sensors together with information on the relationships among them. It is therefore normal to have the sensor take measurements relative to the platform, and depend on the platform for geospatial positioning. This is called a Coordinate Reference Systems (CRS), and is an important part of sensor systems. Every measurement is taken relative to the sensor's CRS.

MEASUREMENTS VS. OBSERVATIONS

The standard distinguishes between measurements and observations. A sensor is set to measure a given property. These measurements results in observations, which can be used immediately or stored for later use. SensorML specifies what is measured, how it is done, and the quality of the measurements. As mentioned above, it does not provide the observation values resulting from the measurements, but it may link to those values.

PROCESSES

Everything in SensorML is modeled as processes. The SensorML process model will be presented here. Information about this model can be found in (Open Geospatial Consortium 2007c). Figure 10 shows an UML diagram of the conceptual process model in SensorML. Every process type is derived from the *AbstractProcess* type, and the difference between physical and non-physical processes is seen by the *ProcessChain*/*System* and *ProcessModel*/*Component* types.

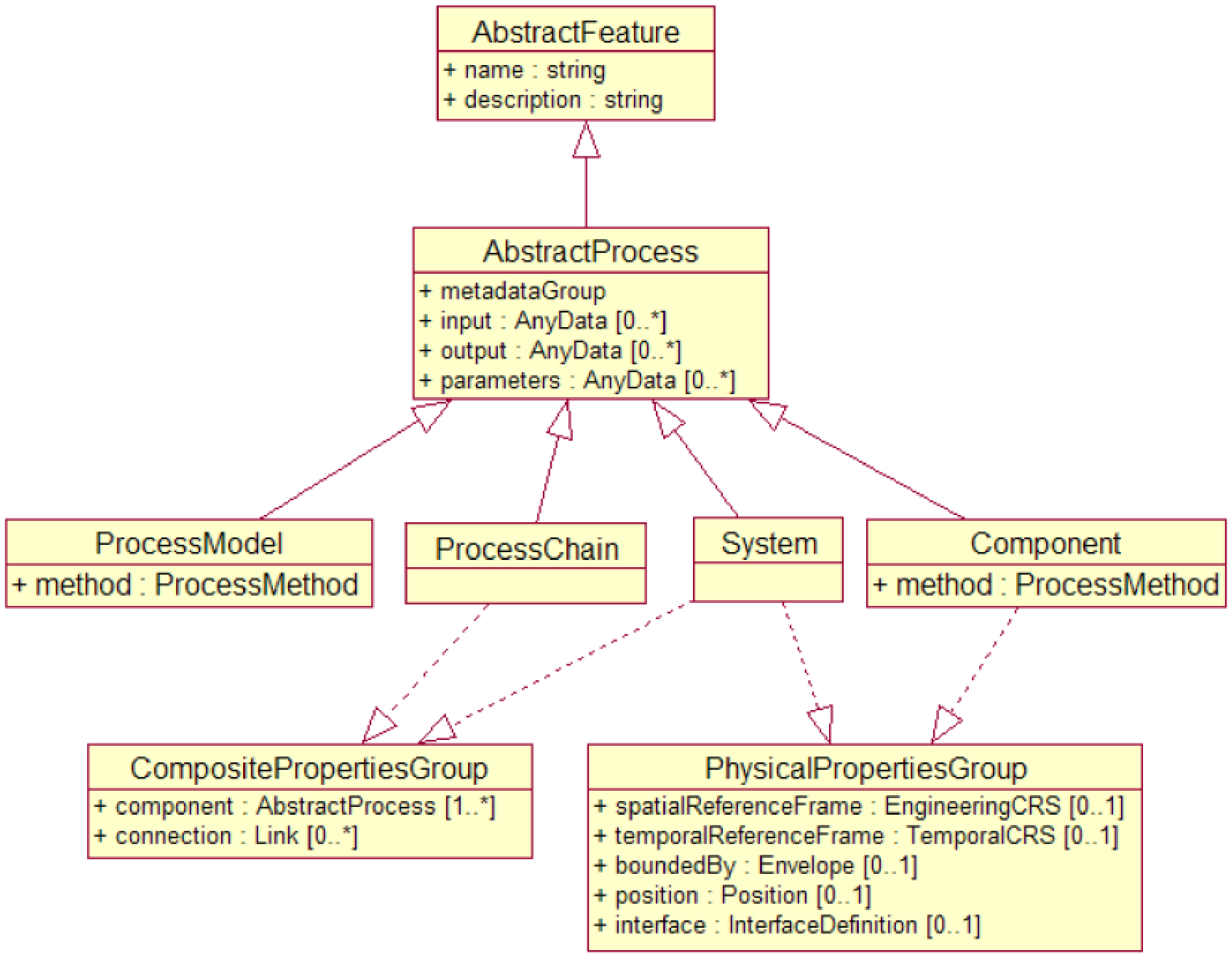

AbstractFeature
+ name : string
+ description : string
AbstractProcess
+ metadataGroup
+ input : AnyData [0..*]
+ output : AnyData [0..*]
+ parameters : AnyData [0..*]
ProcessModel
+ method : ProcessMethod
ProcessChain
System
Component
+ method : ProcessMethod
CompositePropertiesGroup
+ component : AbstractProcess [1..*]
+ connection : Link [0..*]
PhysicalPropertiesGroup
+ spatialReferenceFrame : EngineeringCRS [0..1]
+ temporalReferenceFrame : TemporalCRS [0..1]
+ boundedBy : Envelope [0..1]
+ position : Position [0..1]
+ interface : InterfaceDefinition [0..1]

Figure 10 SensorML process model (Open Geospatial Consortium 2007c)

Sensors are modeled as processes as it is difficult to know where detecting ends and processing begins. Sensors are a combination of detected and processed data. It can also be argued that any detector converts an input signal to an output, and therefore is a process. For the user, this means that there are no differences in how measurements, simulations, or data processing chains are handled, and it is therefore possible to treat everything the same. Figure 11 shows the difference between physical and non-physical processes, as well as atomic and composite processes. These are the four types of processes available in SensorML, in addition to the abstract types. For examples on how such sensor models can be implemented, see the SensorML standard (Open Geospatial Consortium 2007c).

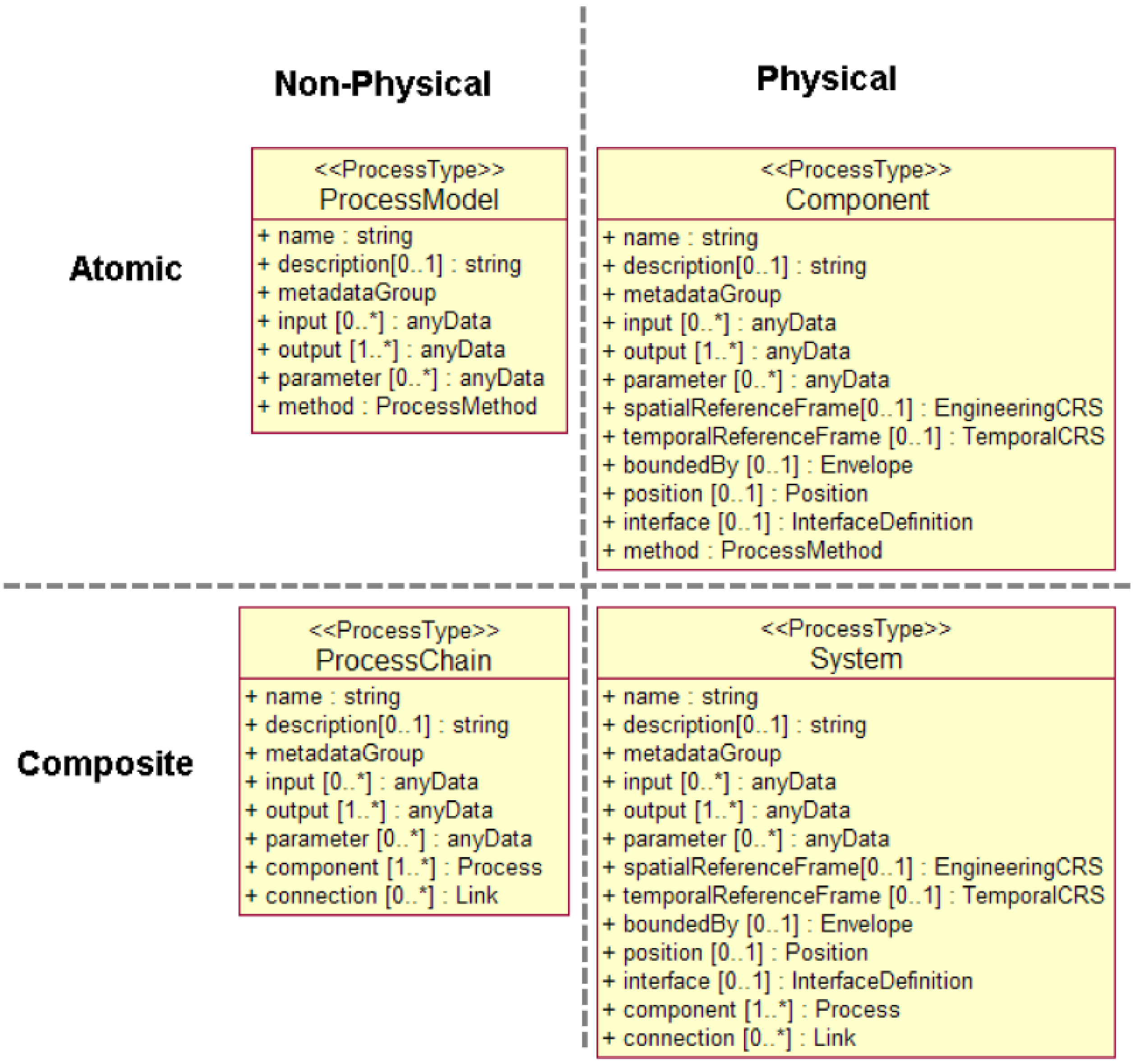

Non-Physical
Physical
Atomic
Composite
<<ProcessType>>
ProcessModel
+ name : string
+ description[0..1] : string
+ metadataGroup
+ input [0..*] : anyData
+ output [1..*] : anyData
+ parameter [0..*] : anyData
+ method : ProcessMethod
<<ProcessType>>
Component
+ name : string
+ description[0..1] : string
+ metadataGroup
+ input [0..*] : anyData
+ output [1..*] : anyData
+ parameter [0..*] : anyData
+ spatialReferenceFrame[0..1] : EngineeringCRS
+ temporalReferenceFrame [0..1] : TemporalCRS
+ boundedBy [0..1] : Envelope
+ position [0..1] : Position
+ interface [0..1] : InterfaceDefinition
+ method : ProcessMethod
<<ProcessType>>
ProcessChain
+ name : string
+ description[0..1] : string
+ metadataGroup
+ input [0..*] : anyData
+ output [1..*] : anyData
+ parameter [0..*] : anyData
+ component [1..*] : Process
+ connection [0..*] : Link
<<ProcessType>>
System
+ name : string
+ description[0..1] : string
+ metadataGroup
+ input [0..*] : anyData
+ output [1..*] : anyData
+ parameter [0..*] : anyData
+ spatialReferenceFrame[0..1] : EngineeringCRS
+ temporalReferenceFrame [0..1] : TemporalCRS
+ boundedBy [0..1] : Envelope
+ position [0..1] : Position
+ interface [0..1] : InterfaceDefinition
+ component [1..*] : Process
+ connection [0..*] : Link

Figure 11 SensorML non-physical/physical processes (Open Geospatial Consortium 2007c)

9.3 SENSOR OBSERVATION SERVICE (SOS)

The Sensor Observation Service (SOS) provides a standard way to access observations from sensors and sensor systems. The SOS specification defines an API for retrieving sensor data and for managing deployed sensors. SOS acts as an intermediary between clients and observation repositories, and it also provides metadata associated with the observations and sensors. The standard separates between two points of view for clients using SOS; observation-centric and sensor-centric. The observation-centric user wants observations from a particular area or a particular phenomenon, but does not know of any sensor providing that information. Sensor-centric users are already aware of one or more sensors, and want to retrieve observations from them. Figure 12 shows a flow chart of how such a process is performed by a sensor data consumer. The service discovery part is done through OGC Catalog Services (CS-W), and is not part of SOS.

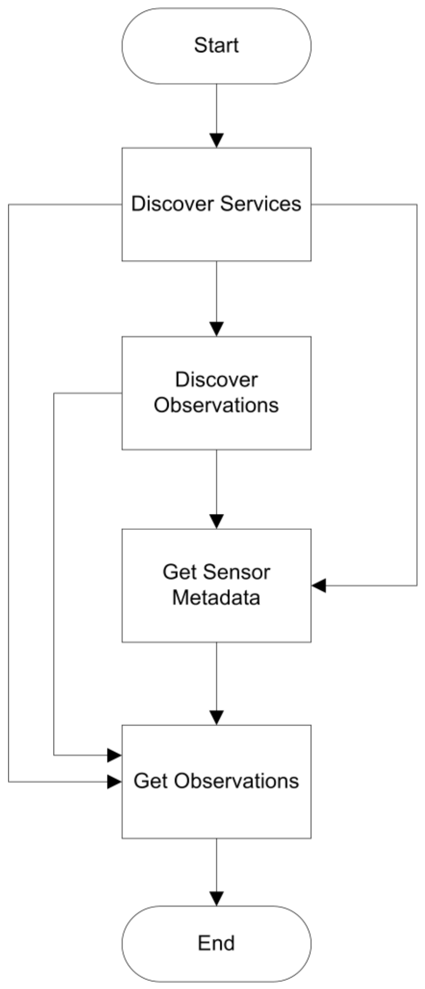


Figure 12 Consumer flow chart (Open Geospatial Consortium 2007e)

The workflow will be similar for producers of sensor data. The only difference is that they insert new sensors or observations instead of retrieving them. The SOS service only allows publishing of observations from sensors already known to the service. If it is a new sensor, that sensor must be registered in the SOS before the observations can be published (Open Geospatial Consortium 2007e) (Botts, Percivall et al. 2007).

The observations provided are sorted into offerings. The SOS is responsible for combining relevant observations into offerings. Two sensors monitoring the same feature of interest would typically be combined into an offering. If these two sensors are taking measurements at different times, they could be provided as two different offerings. This is because a small change in the query could return a very different result. A goal is to have valid queries return as few empty results as possible (Open Geospatial Consortium 2007e).

The Core Operations Profile of SOS has three mandatory methods:

- GetCapabilites() retrieves a list of observation offerings from the service.
- DescribeSensor() takes a URI to a sensor as input and returns a description of the sensor in either SensorML or Transducer Markup Language (TML).
- GetObservations() retrieves observations on the O&M format. The offering containing the observations must be specified, and it is possible to specify sensor (in that offering) and time span of the observation.

The Transaction Operations Profile has two optional methods:

- RegisterSensor()
- InsertObservation()

Finally, the Enhanced Operations Profile has seven optional methods:

- GetObservationById()
- GetResult()
- GetFeatureOfInterest()
- GetFeatureOfInterestTime()
- DescribeFeatureType()
- DescribeObservationType()
- DescribeResultModel()

9.4 SENSOR PLANNING SERVICE (SPS)

The Sensor Planning Service (SPS) is developed for clients to determine collection feasibility and to submit collection requests to one or more sensors or platforms. It defines interfaces to determine feasibility of planning requests, submitting planning requests, retrieve status of requests, and to update or cancel requests. The standard defines the users of SPS as large enterprises wanting to automate complex information flows (Botts, Percivall et al. 2007). SPS is created to support collection management, which means coordinating resources involved when collecting information. There are three subsets of collection management: requirements management, mission management, and asset management (Open Geospatial Consortium 2007d).

The SPS operations can be divided into two parts, information and functional operations. The current standard specifies eight operations. The informal operations are (Open Geospatial Consortium 2007d):

- GetCapabilities
- DescribeTasking
- DescribeResultAccess
- GetStatus

The current functional operations are:

- GetFeasibility
- Submit
- Cancel
- Update

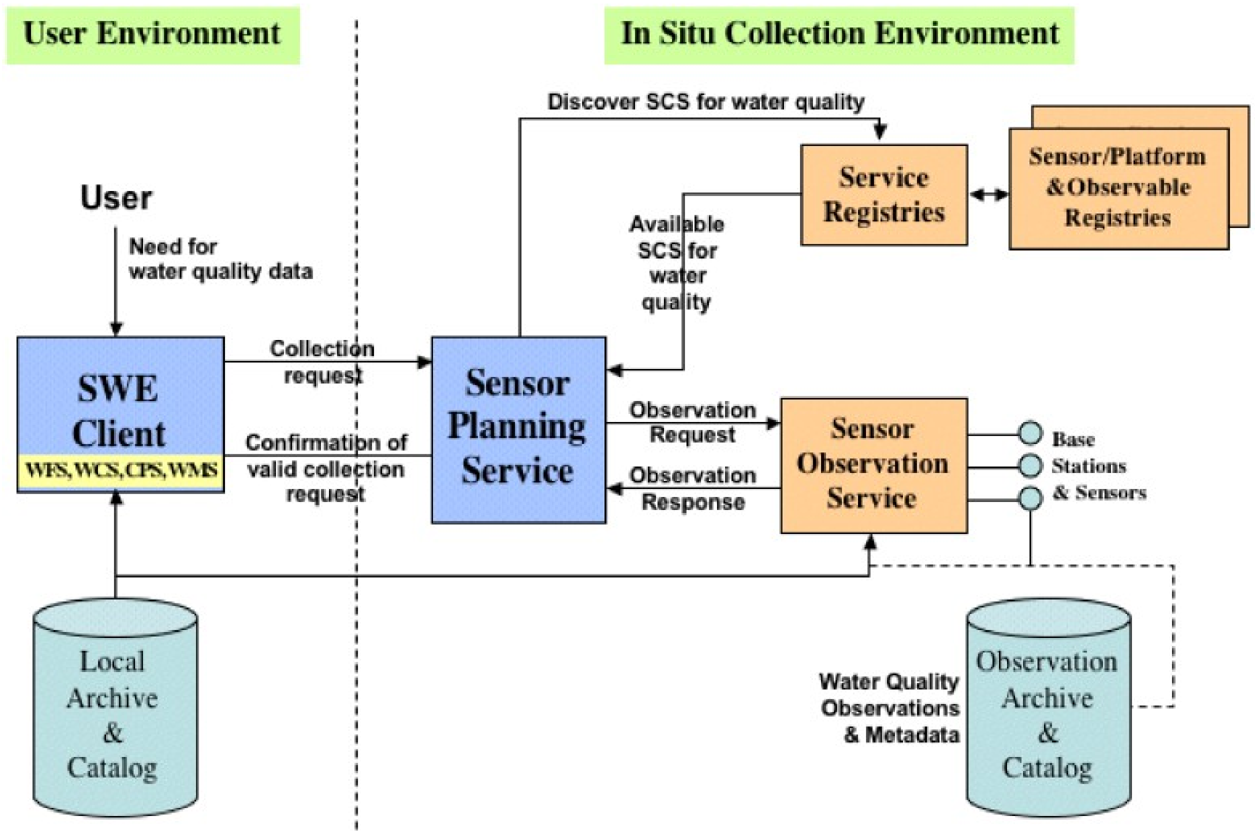

User Environment
In Situ Collection Environment
Discover SCS for water quality
Service Registries
Sensor/Platform & Observable Registries
User
Need for water quality data
Available SCS for water quality
SWE Client
WFS, WCS, CPS, WMS
Collection request
Confirmation of valid collection request
Sensor Planning Service
Observation Request
Observation Response
Sensor Observation Service
Base Stations & Sensors
Local Archive & Catalog
Water Quality Observations & Metadata
Observation Archive & Catalog

Figure 13 SPS overview (Botts, Percivall et al. 2007)

Figure 13 shows how a Sensor Planning Service fits in an environment with in-situ sensors and how it communicates with the Sensor Observation Service described above.

Registries play an important role in the SWE, and an operational system will have many registries for sensors, sensor types, and observations. Figure 14 shows the different roles registries play in a sensor web. The left side contains registries for sensor and observation types, while the right has actual sensors and observations. It also shows the roles of SensorML and O&M.

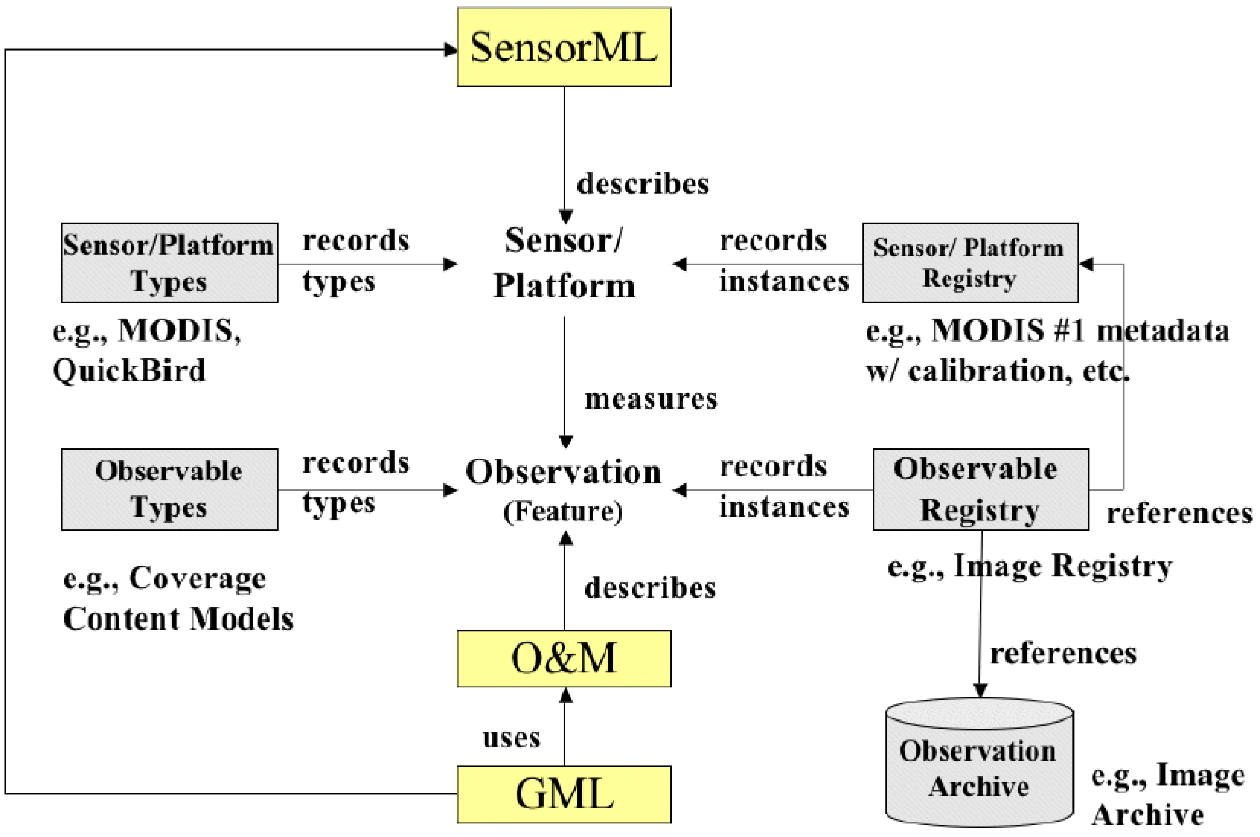


Figure 14 Registries in SWE (Botts, Percivall et al. 2007)

9.5 OTHER STANDARDS

This section will here briefly describe the remaining three standards in the SWE. The Transducer Markup Language (TML) is used to describe transducers and transducer systems. It supports capturing, exchanging, and archiving live, historical, and future data. TML supports streaming real-time observations from sensor systems that are time tagged and sensor-references, while SensorML describes the systems models to help interpret the streaming observations (Botts, Percivall et al. 2007).

Sensor Alert Service (SAS) provides interfaces to an alert service, providing information on capabilities, protocols, and subscription. SAS is more like a registry than an alert system. Sensors and data providers register to a message server, which then forwards the information to SAS. Users will then lookup in SAS to retrieve the communication endpoint for the messaging server. It is therefore necessary to use other alert protocols and standards not specified in SAS (Botts, Percivall et al. 2007).

The final standard is the Web Notification Service (WNS). This is an interface for a service where clients can conduct asynchronous message interchange with different services (Botts, Percivall et al. 2007).

This was the final chapter of Part II, and concludes the background study of this thesis. Part III presents the contributions of this thesis.

Part III Own Contribution

10 INTRODUCTION

This part describes the thesis' contribution. It first presents an evaluation of the EPC Architecture Framework and the Sensor Web Enablement. They were evaluated with regard to their fitness of use for item tracking and context information. After this, a chapter with a stakeholder analysis is presented, listing important stakeholders for a system for item tracking and context information. This analysis was then used to develop a set of scenarios presented in Chapter 13. These scenarios present potential uses for a system combining item tracking and context information.

The stakeholder analysis and scenarios were used as a basis for a requirements specification of a software system, presented in Chapter 14. Based on these requirements a software architecture which could be used to build a system enabling the scenarios was designed. This architecture combines the two frameworks, and is presented in Chapter 15.

OUTLINE

The rest of this part is organized as follows:

- **Chapter 11: "Evaluation and Comparison of EPC AF and SWE"**
- **Chapter 12: "Stakeholder Analysis"**
- **Chapter 13: "Scenarios"**
- **Chapter 14: "Requirements Specification"**
- **Chapter 15: "Architectural Solution"**

11 EVALUATION AND COMPARISON OF EPC AF AND SWE

This chapter evaluates the EPC Architecture Framework (EPC AF) and the Software Web Enablement (SWE) with regards to tracking of items and context information, and presents opportunities for how they can be integrated. Finally, the two frameworks are briefly compared.

11.1 EPC ARCHITECTURE FRAMEWORK

EPC Architecture Framework is a collection of standards and core services operated by EPCglobal with the goal of using Electronic Product Code (EPC) to enhance the supply chain (EPCglobal 2007c). The framework has standards for both hardware and software, from the coding of RFID tags to sharing of EPCIS events. This thesis concentrates mostly on the EPCIS part and the sharing of information. For an introduction to EPCIS, see Appendix B. The specified user memory of the tags is capable of storing sensor observations, and this can be a good solution for RFID tags with attached sensors. The benefit of also providing hardware specifications is that it ensures interoperability between different vendors. This also applies to the software level, as software created by different developers will be able to interoperate.

EPCIS FOR ITEM TRACKING

EPCIS is capable of describing events related to items tagged with EPCs, and provides interfaces for sharing these events. It basically provides everything needed related to tracking of items and sharing this information with business partners. The EPCIS information is described as the *“what, where, when, and why of items”*, meaning that it has information on what an item is, where it has been, when it was there, and why it was there.

One benefit of using this standard is that it does not require the use of RFIDs to carry the EPCs; the information can come from any source, e.g. barcodes. This means that items not tagged with RFID can still be a part of solutions based on EPCIS as long as they are identified by an EPC. This is a huge benefit and makes it much easier to start using EPCIS than if every item needs an RFID tag and every part of the value chain needs RFID readers installed.

Other benefits of using this standard are that it is an open standard available to everyone, and that the use of RFID and EPC is starting to spread. If an organization or industry has already standardized on EPC, it will be easier for them to adapt to solutions based on EPCIS than if something else were used for item tracking. Large software companies have already started implementing solutions based on EPCIS; among them are SAP, IBM, and Oracle. EPCIS is also very well documented, and the needs of different industries have been taken into consideration. The standard is easily extendible, and can be adapted to the needs of specific industries or to groups of trading partners.

A drawback with the EPC AF and EPCIS is that not every part is finished. The standard specifies an EPCIS Discovery Service, as well as a centralized authentication service, but none of these are finished. This should not be a big problem, as they are under development and it is possible, in the meantime, to develop substitutes. These missing services are especially important for large EPC networks between many organizations, but such networks will probably not be implemented for some years.

There are many pilot implementations but few full scale implementations of EPC AF and EPCIS in large organizations. A main reason for this is that the standard is very new; EPCIS 1.0 was released April 12, 2007, and EPC Architecture Framework 1.0 at July 1, 2005.

EPCIS FOR CONTEXT INFORMATION

The EPC AF and EPCIS standards handles the area of item tracking very well, but has little or no support for context information. EPCIS events have the possibility to include some context information, but this information is related to business processes such as business step, related transactions, disposition of the items etc. In e.g. the fresh food industry, this will not be sufficient.

The EPC Architecture Framework focuses on extensions, and it is possible to extend the EPCIS events with additional information. This could be a way to include sensor data and other context information. It is possible to create an event that is a subclass of the EPCIS Event called, e.g., ContextEvent, containing context information. By this approach the ordinary EPCIS system can be used and the only requirement would be that the *EPCIS Capturing Application* is extended with information about the new ContextEvent and that sensor data is sent through the *EPCIS Capturing Application* to generate the events.

The problem with this approach is that there are no descriptions of how the observations shall be encoded, or how information about the sensors used shall be provided. This would have to be provided by some other means. SensorML can be used for sensor descriptions and the observation results can be encoded in O&M (Observations & Measurements). These two standards would give the needed information, but it would be problematic to tie sensors to EPCs, especially if they are not physically on the same item at all times. It would also require a means to publish and retrieve this information. The standard *EPCIS Query Interface* only support predefined queries, and context-related queries would be problematic.

An easy solution would be to just extend the EPCIS Event base type with a context-field providing the observation results, and leave it up to the user to figure out how to understand the data. This could fit into ordinary EPCIS systems, and there would be fewer problems with queries as the user can use the same predefined queries as for ordinary EPCIS operations. It would however not be possible to specify context-related parameters in the query, e.g. a search for all items with too high temperatures. The only difference from this new event type and a standard EPCIS system would be that the resulting events contain an additional context-field. The problem would of course be how to encode this field. For very simple observations, this is no problem. If for instance the fresh food industry wanted to know the temperature of the goods, they could include a field called *TemperatureInCelcius* containing an Integer with the value. This approach is mentioned in the EPCIS standard, but it limits the sensor types used, as it would require a separate field for each sensor type.

It would also be difficult with more advanced observations. As mentioned in Part II, it is not enough to have observation results without knowledge of how the observations has been measured and retrieved. Information about the sensor or sensors used is also important. It is necessary to know such parameters as quality of the measurements, error rate of the sensors, measurement locations, sampling rates etc. This could of course be encoded in every EPCIS Event, but it would be a huge overhead and the user memory of the RFID tags is of limited size. The only reasonable way would probably be to have some kind of sensor repository with sensor information, and it could be possible to include a link to this in the EPCIS event.

None of these solutions are very good, and some kind of external system would be needed in any case to provide descriptions of sensors and information on how to interpret the observations. Even if this was available, it seems cumbersome to force EPCIS into doing something it is not really suited for. Including context information in EPCIS can be done, but it is probably not an optimal solution. The principle of separation of concerns also applies here. EPCIS is good at item tracking, but it is originally not made for sensor and context information, which thus should be left to something else.

The conclusion is that EPC AF is a very well developed standard, and that it is already starting to get a wide use for item tracking, especially in the retail industry. It is also possible to support context information in EPC AF, but this would be a limited support with many problems.

11.2 SENSOR WEB ENABLEMENT

The Sensor Web Enablement (SWE) is, as mentioned in Chapter 9, a framework of open standards created to support a sensor web. A sensor web is a sensor network available on the web using standard protocols. SWE contains many different standards to support this, focusing on software. Hardware standards are not included, and this means that there are no guidelines as to what kind of sensors can be used or how they are read; this is up to the developer. What SWE do provide is SensorML and TML used to describe sensors and transducers, and Observations and Measurements (O&M) used to describe the observations. SensorML/TML descriptions will typically be provided by the manufacturer together with O&M templates for describing the results. Software developers will need to somehow obtain these models, enabling them to understand how the sensors are functioning and how the output should be encoded. The rest of this thesis will focus on SensorML, and not TML, because the sensors relevant for this thesis would be described by SensorML.

The benefit of the approach with only standards for software is that sensor manufacturers are free to create the sensors as they want. There are no restrictions on how the sensors should function or how they should be read, and the SensorML specifications are created so that it should be possible to create models for any sensor. The drawback is that there are no assurances of interoperability between different hardware vendors, which is one of the benefits of the EPC Architecture Framework. The observations need to be obtained from the sensors, and a common standard for this would be useful. A possible reason for not including this in the framework is that sensors varies much more in functionality and type than e.g. RFID tags and readers, and it would be difficult to provide a common standard with enough functionality. This lacking standardization is no problem for this thesis, as the focus is on what happens after the sensors are described by SensorML and the results are encoded in O&M.

SWE FOR ITEM TRACKING

The Sensor Web Enablement framework does not describe any support for item tracking, but it could however be possible to achieve. This could be accomplished by describing RFID readers in SensorML and including the identity of the tracked item, for example an EPC, in the observation. The observation result could be encoded in O&M, and would include the time of the reading. The location of the item would then be obtained from the SensorML model of the reader.

Although this is possible there are many problems with this approach. It would mean forcing SWE to do something it is not meant for. Using RFID tagged items as an example; it would probably be difficult to get every manufacturer of RFID readers to write SensorML descriptions of the readers. It would also be problematic to get other organizations to use this solution as it is not an industry standard. At every point in the supply chain where the items are detected it would be necessary to have SensorML models of the reader as well as creating O&M encoded observations.

Finding information in such a system would also be problematic, and especially when sharing between organizations. Item tracking solutions should focus on items, but SWE focuses on sensors, sensors systems, and observation offerings from these. It would also lack functionality to push events from the system to users, which is possible with, e.g., EPCIS. It would be possible to extend the existing interfaces with methods providing this functionality, but the framework does not specify where or how extensions should be made. This means that there are no assurances that extensions will continue to work with future versions of the framework, which especially is a big problem when sharing information across organizations. It would require all participating organizations to use the same version of the framework, or at least versions compatible with the extensions, and it would be difficult to upgrade as new versions are released.

Although it is clearly possible to use this approach, it is not an optimal solution. The biggest problem would probably be to get organizations to use this solution instead of already established standards. It is likely that many organizations, especially within retail, will first implement an item tracking solution and later extend with context information, not the other way around. This means that it would probably be better to use an established standard for item tracking. An established standard would enable organizations already using item tracking based on this standard to extend their existing solutions with context information, instead of implementing an entirely new solution using SWE for item tracking.

SWE FOR CONTEXT INFORMATION

The standards found in the Sensor Web Enablement framework support description of sensors, description of observation results, services to obtain these data, and much more. The SensorML models are much more advanced than what is needed for this thesis, and the O&M standard is capable of describing a huge range of observation types not needed by, e.g., the fresh food industry. It is for example possible to describe remote sensors on a satellite monitoring conditions on the earth, which is a bit more advanced than a temperature sensor mounted on an RFID tag. But it also provides what is needed for this thesis, and it meets every requirement for context information.

Using the Sensor Observation Service (SOS) it is possible to retrieve sensor descriptions, providing information about where the sensor is located, how it is configured, what kind of observations it provides, and the time period it has been operating. There are also methods to retrieve observation results, which can be limited to a particular sensor or to observation offerings. Observation offerings are a collection of sensors providing related information, i.e. a temperature sensor and a humidity sensor located in the same storage area. The results can also be time limited, helping the user to retrieve only relevant data.

The one thing SWE does not provide is a link between items and context. SWE is a sensor-oriented framework, and it is not possible to query the system to find context information related to an item. It can also be observation-oriented, but it is still related to a specific area or a particular phenomenon, such as a storage facility, and not to an item. It does not know which items that have been in a particular area. This means that it provides the necessary context information, but it does not relate that information to particular items. For sensors fastened directly on the item, e.g. RFID tags with temperature sensors, this is no problem. But it limits the possibility of retrieving additional information from sensors not directly related to the item, such as sensors in a truck or in a storage facility. A separate system is needed to keep track of items related to sensors. If it is possible to determine where an item has been, SWE can give sensor observations for those locations.

One problem using this framework can be support of the standards. SensorML has been under development for a long time, but the OpenGIS version of SensorML was released as late as July 2007. The rest of the standards in the framework are also very new. It will take some time before all sensors have descriptions in SensorML, and for enterprise systems to support reading these formats. However, implementing support for these standards would not be very difficult since XML is used as the message format.

The conclusion is that SWE is very well suited for context information, and that it is possible but not optimal to use it for item tracking. The standards in the framework are new, and they show signs of not being completely finished. Even though the standards are new, they are very extensive, and show a great potential for wide use in the future.

11.3 INTEGRATION OF EPC AF AND SWE

It is obvious that a solution integrating these two frameworks would provide everything needed to extend an item tracking system with context information. EPCIS in the EPC AF is good at tracking items and providing related event information to every part of a value chain, and SWE is good at providing sensor information and observation values. There are mainly three ways to integrate these systems: extend EPCIS with SWE, extend SWE with EPCIS, and create a third system which uses both EPCIS and SWE.

Extending EPCIS with SWE has briefly been discussed above. Focusing on the Sensor Observation Service to provide sensor information, it is possible to extend EPCIS events with links to relevant SOS systems. This will provide users aware of the extension with useful information, and other users can continue to use EPCIS as before. This will require the EPCIS capturing applications to know which SOS systems have relevant information for the EPCs it generates events for. This means that if an item has passed through several parts of a value chain, each with its own capturing application and EPCIS repository, each of these needs to be checked for potential context information. Including links to SOS services in the EPCIS events is probably a better solution than including the observations themselves, which have many problems as discussed above. Including this link would however break the principle of separation of concerns, as it includes context information in a tracking event.

Another possibility is to extend SWE with EPCIS information. This is probably not a good solution, as the SWE framework has no focus on extensibility, and it would give problems with interoperability as discussed above. This solution would mean that each time a sensor generates an observation it must check for EPCs related to that observation, and if needed either generate EPCIS events or pass the information to an *EPCIS Capturing Application*. It would also be impossible to query the system for item information, as there are no such interfaces in the SWE standard. A third party application would be needed to tie EPCs to sensors. It could be possible to offer an EPCIS interface from e.g. the Sensor Observation

Service or the Sensor Planning Service, but this would as mentioned be a non-standard extension, and it could be difficult to get organizations to adopt such a solution.

The final option is to have EPCIS and SWE as separate systems, creating a common interface which combines the functionality of these. In this way, users only interested in either EPCIS events or sensor observations can use the existing systems as before, while users interested in both can use the new interface. This would require a bit more custom implementations than the two solutions above, it would e.g. be necessary to have a mapping between EPCs and sensors to know which sensors has information on which EPCs. The benefit of this solution is that it keeps the original frameworks as they are, without modifications. It also enables the frameworks to evolve on their own, and new version of either framework would probably not require much update to other parts of the system. The drawback is that a third system would have to be developed and maintained.

11.4 COMPARISON OF EPC AF AND SWE

Even though these frameworks concern different domains, it is interesting to compare them. EPC Architecture Framework tries to provide all necessary functionality for an item tracking solution, from the hardware level with RFIDs carrying EPC tags up to global software services supporting the EPC network. Software Web Enablement similarly tries to provide everything needed for a sensor web, from sensor and observation descriptions to providing this information through services. The main difference here is that SWE does not consider hardware at all.

One thing noticed when reading these standards, is that they seem to be of differing qualities. The standards from EPCglobal seem very well documented, are rather easy to get into, and they seem ready for implementation. It is clearly defined which parts are finished, and what remains for future versions. They have also taken into consideration that the standards might be used in many different industries, and the standard documents clearly specify how the standards can be made to fit these industries.

This is not the case for the standards in SWE. It is not obvious what information is found in the different chapters of each standard, and the same information is explained in different ways across the standards in the framework. SWE also lacks a central standard combining all the others. The SWE framework consists of many separate standards, but the framework itself is only described by a high level architecture document (Botts, Percivall et al. 2007). This document is an introduction to the various standards, but it would be better if the framework had tied the standards tighter together. This would not necessarily have an impact on how the various standards develop, but it would help organizations wanting to implement a solution based on the framework.

The standards in SWE are also clearly in version 1, as it is written several places in the documentations that a particular section should be revised, or that the approach taken in a situation should be changed in a future version. The EPCIS standard is also in its first edition, but it does not have the same problems. The cores of the SWE standards seem to be very good, and have a lot of potential, but they will benefit from some revision. This might also be a reason why not many commercial uses of SWE are found.

These two frameworks come from two very different sources, and this is probably a major reason for the differences. EPCglobal is a global standards organization, but it is initialized by GS1, and it clearly has a commercial purpose. They focus on getting the standards adopted by as many industries as possible, which is reflected in the standards through their focus on extensibility. This is of course very important, as the potential of an EPC tracking system is greatest when every part of the value chain is using it. It is also clear that EPCglobal wants organizations to use EPC for identification and the EPC Class 1 Gen 2 UHF standard for RFID tags, even though there are many other standards for RFID tags, e.g. various ISO standards. This is of course because EPCglobal is behind the various EPC standards, but this does not mean that they are the best in every situation.

This is in contrast to the Open Geospatial Consortium (OGC), which has released the SWE framework. OGC is an international industry consortium working on publicly available interface specifications through a consensus process. Their process starts with an interoperability problem, and then they try to develop standards solving this problem. At the end they present a specification describing how this should be solved, and the specification can be adopted by anyone as it is made freely available on the Internet. This is very different from the EPC Architecture Framework. EPCglobal has, through its members, created a framework that will enhance the supply chain through the use of EPC. Each standard in the framework is focusing on this goal, and they clearly complement each other. OGC has many standards that are not directly related as there is no common framework surrounding them. The SWE framework does not yet have a separate specification, only a White Paper from OGC describing a high level architecture (Botts, Percivall et al. 2007). OGC also has other standards that are not related to SWE. These differences between EPC AF and SWE does not mean that one framework is better or worse, it only illustrates the different approaches taken by the two organizations (EPCglobal 2008b) (Open Geospatial Consortium 2008).

12 STAKEHOLDER ANALYSIS

To better understand what is required by a software system, stakeholders need to be analyzed. This chapter will present different stakeholders and analyze their interests in the system. The stakeholders and their interests define the requirements at an abstract level. This will lay the foundation for later sections, exploring possible scenarios and a related requirements specification.

12.1 GOVERNMENT

The Norwegian government has recently given control of fresh food a considerable amount of interest. This recent interest is due to food accidents involving human deaths and injuries. Contaminated food was sold from stores, and when the contamination was detected and investigated no immediate source could be found. They spent several days tracing the source before it was found. This event and other recent events call for solutions giving better control, with tracking of location and environment data. Their interest will be in standards to be used by the whole industry, sharing of information between enterprises and from enterprises to the government, and privacy and safety issues.

12.2 CONSUMERS

Fresh food and other goods tracked by an item tracking system will at the end arrive at the consumers, and they will therefore be an important stakeholder. Consumer interests are in safety and privacy issues, and also in easiness of application, in how well this new solution can be used by them. Their interests may not come at first, when retailers start using this system, but later on when stores start to share information from the system as a service to the customer.

Consumers are skeptical to new systems, and these days tracking has a bad reputation. This is why their main interests are safety and privacy, but if these are met, usability and usefulness of services become important.

12.3 PRODUCT MANUFACTURERS

The manufacturers of products will be greatly influenced by a tracking solution, and are therefore an important stakeholder. Their interests would be cost, security, privacy, sharing of information, and standards.

A tracking solution relies on manufacturers to use EPC for their products, so they can be tracked later in the supply chain. Descriptions of goods are also important, and a standard method of retrieving this information. Therefore, standards for information sharing are important to manufacturers. Because the products are to be attached with an EPC tag, the price of this addition is important, because extra costs mean less income. As always, with sensitive information, security and privacy are important.

12.4 TRANSPORTATION COMPANIES

Goods are transported and companies doing this job are important stakeholders. Their interests will be safety, sharing of information, reliability, and trust.

The transportation companies have to be sure that information about how they treat products is reliable and not altered after they deliver them. They need to share this information with their customers, and they need customers to trust the correctness of this information.

12.5 RETAILERS

Retailers buying goods before they are sold to consumers, sit at the end of the value chain or in other words the last station before consummation. They are therefore important stakeholders with a lot of interests, such as: cost, increase of product costs for example; security, of information; privacy, mainly concerned with consumers; sharing of information, concerning other enterprises and government; and standards, to be supported and widely known.

Retailers need control over their value chain, tracking and quality assurance. To be able to prove quality through every part of the value chain, and give reassurance of this.

12.6 DEVELOPERS

Companies or individuals required to build information systems in this domain will be interested in best practices. Looking at suggested solutions will influence their choices, and help them build their system more effective.

13 SCENARIOS

This chapter presents several scenarios for future use of wireless sensors based on the stakeholder analysis performed previous chapter. The scenarios present future applications and investigate data needs. What a sensor tracks is not important to the sensor itself, but to the systems using the recorded data. The scenarios are grouped by topic, and each topic is explained before different scenarios are presented for the given topic. Every scenario is then further explained. The scenarios are numbered hierarchical in topic and scenario, e.g. topic 2 scenario 3 is numbered SC 2.3.

13.1 QUALITY DETERIORATION (SC 1)

Fresh food, but also food in general, has an expiration date. The quality of the food decreases as time passes, but most important is how it has been treated during its lifetime. Today, expiration dates are set by the government and by law, different for every type of goods. A good example of why this is not a perfect system is the expiration dates for eggs in Norway. Every Easter, eggs have longer durability. Not due to anything else than a petition from the industry to the government.

The expiration dates are set based on general rules of safety, meaning that most food would actually be fresh long after the expiration date. The environment in which the food has been stored and managed is often the real reason of quality deterioration.

13.1.1 AUTOMATIC AND DYNAMIC EXPIRATION DATES (SC 1.1)

Expiration dates can be explained as a function of time and environment. Important environment properties are temperature, humidity, light, and exposure to possible contamination through contact with equipment or human hands. A biological research into these properties would enable a function to be implemented in a small computer, a different function for each product type.

Although there are several factors to food quality, the most important factor is temperature. The increase in temperature will have a high impact on the quality deterioration over time. A function for time and temperature, given parameters for how the food type reacts to temperature, is (Brock 2003):

$$Q(t) = \frac{25}{432e^{-\frac{15034}{5T}}t + \frac{1}{4}}$$

T = Temperature in Kelvin degrees

t = Time in days

Q(0) = 100% quality of product

This function would be different for different types of food, because of different input parameters. For every type of food a similar function could be calculated.

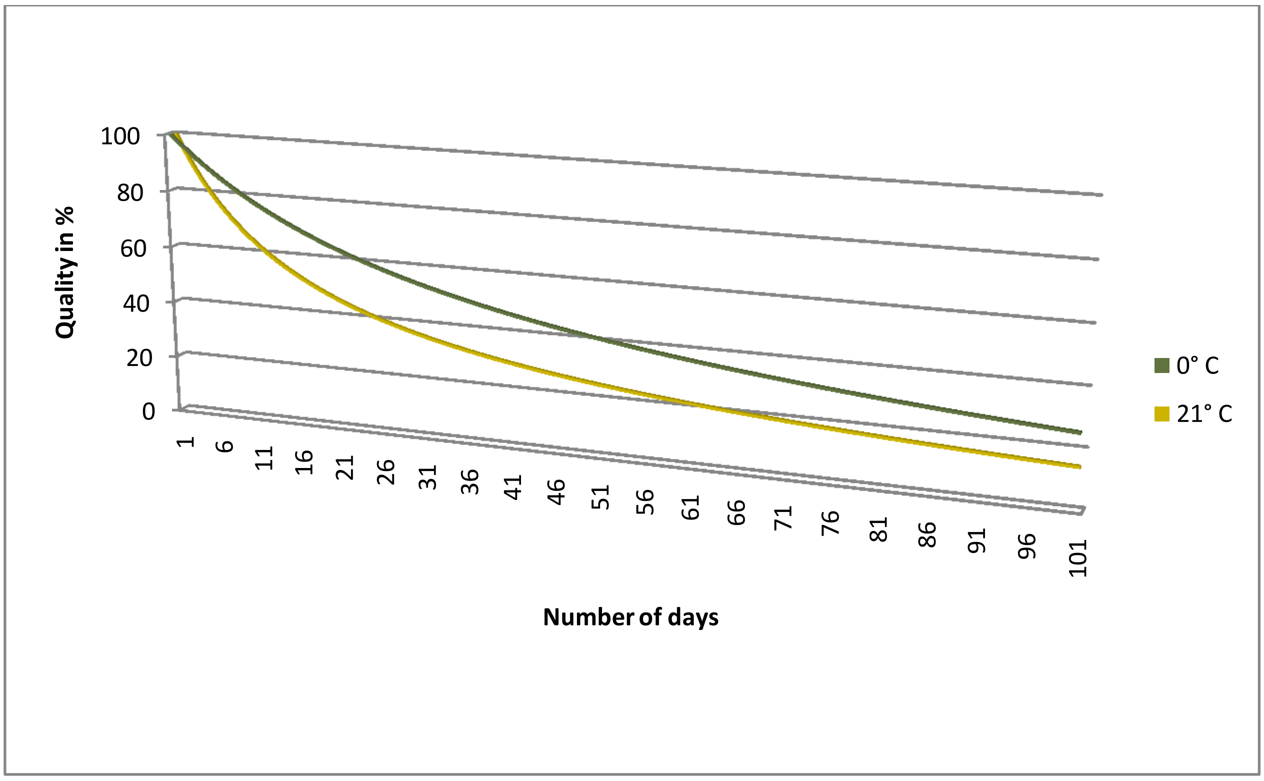


Figure 15 Deterioration of one type of food at two different temperatures

Calculation of expiration date based on treatment of the product over time, and not just a guess or prediction, enables creation of dynamic expiration dates. An example of use could be milk cartons, where the amount of time outside of the refrigerator would have an impact on the expiration date. If the carton is left too long outside, it will go sour before the expected expiration date.

13.1.2 RELOADING OF SHIPMENT BASED ON QUALITY (SC 1.2)

Transportation of fresh food over long distances can be a challenge. Fish from the north of Norway can travel several hundred kilometers by trailer trucks. Failures of cooling systems or loading/unloading of cargo can deteriorate the quality of the fish. Being able to detect this deterioration and allowing a reloading of the goods along the route will minimize the impact. Fresh fish close to contamination can be reloaded midway and sold or worked up into new products. This will prevent fresh fish to be sold in stores after contamination or at a lower quality than expected or ordered.

13.1.3 PROVING THE QUALITY OF FRESH FOOD (SC 1.3)

Fresh food has not been frozen, and one can clearly distinguish between the two when it comes to the quality of the food. Therefore, consumers often pay a higher price for fresh food. Fresh food that has been frozen should no longer be sold as fresh food. Tracking of temperature over time would enable the consumer to check the quality of their product, e.g. by checking that defrosted products are marked and not sold as fresh food.

13.2 TEMPERATURE PROFILES (SC 2)

Tracking the temperature of goods in containers may reveal information about the conditions inside the container if all the sensors' data are combined. Tracking this over time and building a profile of temperatures inside the whole container would reveal a unique behavior for the given container. To be able to track this information, it must be possible to determine the sensors' positions relative to each other, defining a sensor area. This information can then be used to visualize the environment and change where different goods are transported and managed. The data needed is time and temperature, together with the sensor locations.

13.2.1 HOW MANY SENSORS ARE NEEDED IN A GIVEN SPACE (SC 2.1)

For a given space, e.g. a container, temperature profile information could be used to reveal the exact need for sensors. If the temperature is identical in the whole container, only one working sensor is needed. If trailers are considered, transporting goods in a temperature-controlled box, loading and unloading of goods will affect the temperature differently in the front and the far back. Goods with high thermal capacity would also affect an area if the temperatures are changing. Mapping this information for different types of goods transported in different environments, will give information about how many sensors are needed for every given scenario of transportation done in the past.

13.2.2 DETECTION OF COLD SPOTS AND WARM SPOTS (SC 2.2)

Using the same example as in SC 2.1, a trailer with a temperature-controlled box, information about the temperature profile could be used to detect cold spots and warm spots. This information would reveal where trailers have spots with poor or broken insulation. This information could also be used when placing the goods for transportation. Placing goods right next to a cooler might not give the same conditions as being placed at the far corner of the box. One important issue here would be how the cooler is regulating the temperature, whether it is an air cooler or surface cooler.

13.2.3 MAPPING THE NEED FOR INSULATION OF GOODS UNDER TRANSPORTATION (SC 2.3)

During transportation of goods, unloading and loading is bound to occur. This will affect the condition and temperatures of the goods. Placing two sensors, one inside and one outside the box, a temperature profile of the differences between the two could be detected. Tracked over time this information would reveal the differences in temperature and for how long they occur. This information can then be used to calculate how much insulation is needed. An example could be transportation of fish, using ice as insulation. Ice is heavy, and the less needed the better because of the cost of transporting unnecessary weight. Tracking several transportations in a supply chain for fish, the amount of ice needed could be optimized. This would decrease cost, but not the quality of the fish transported.

13.3 SENSOR COLLABORATION AND INTELLIGENT GOODS (SC 3)

Sensors are already used to control different environment controlling devices, e.g. cooler systems in trailers. When all goods transported by a trailer with a cooler have a sensor, the goods' sensors could be used instead. The trailer's sensor network can be outdated, and by using the goods' sensors correct information are retrieved. Whenever the cooler system needs information about the temperature, it can use the goods' sensors if its own sensors are not working properly. This can also be reversed, and the goods can use sensors already inside the trailer to retrieve information about their environment. This way the trailer can optimize its sensors, and one only needs to know which goods are on the trailer at a given time. Both scenarios take into consideration where to put the trust, and this depends on each given scenario. This makes both solutions equally valuable, in different settings.

Sensors used for tracking the conditions of goods can be semi-passive or active. Semi-passive sensors only track the conditions and save them for later, to when readers retrieve the information. Active sensors can act or communicate at given intervals or conditions, enabling the goods to make decisions and control other devices influencing their environment.

13.3.1 AUTOMATIC ALARMS OF VIOLATIONS (SC 3.1)

Each ware or every type of ware in transportation may have quite different requirements to their environmental conditions. The example with trailers transporting goods in temperature-controlled boxes, is a good example of this if extended to include transportation of different types of goods for every transportation or at least changing from time to time. The goods may be able to communicate themselves whenever the demands of the environment are not met. Each ware would have a sensor with a programmable minimum and maximum temperature, communicating whenever these conditions are broken. In this way the driver or receiver would know, and measures could be taken to avoid contamination of the food or further quality deterioration.

13.3.2 AUTOMATIC ENVIRONMENT CONTROL (SC 3.2)

By communication between the sensors and devices controlling the environment, sensors could be used not only to detect the conditions of the environment, but also controlling it. In the example used so many times before, trailers with temperature-controlled boxes, the trailers have their own sensors controlling temperature. Instead of only alerting the surroundings and systems whenever requirements are broken, the goods could control the cooler to lower or raise the temperature. This way, the cooler would automatically satisfy the needs of the goods transported.

13.3.3 COLLABORATION OF GOODS (SC 3.3)

For transportation of goods of different types, minimum and maximum temperatures can be set based on data about the different goods. This can be done automatically by communication between the sensors and based on their own and neighbors' demands. These minimums and maximums could then be communicated to cooler devices, or the sensors may only communicate warnings whenever the environment does not meet the demands. One issue here would be disagreement of minimum and maximum temperature, if the allowed span for every ware in the transport does not overlap. Another issue could be the placement inside the compartment. Goods placed right next to the cooler outlet might disagree with other goods when it comes to minimum and maximum temperature. They might say it is too cold, and ask for an increase in temperature, when other goods at the far end might ask for a decrease in temperature. This could be solved by the sensors themselves, one agreeing to break their conditions, preferably the ware with the least impact on quality if the condition is not upheld.

13.4 HIERARCHY OF GOODS WITH SENSORS (SC 4)

Transportation of goods is done in several ways, but some aspects apply in every case. Goods can be placed in boxes, and boxes can be placed on pallets. Every pallet can then be loaded on a trailer or in a container. This natural hierarchical composition of goods can be used to minimize the need for sensors and readings. Placing only a few sensors at the top of the hierarchy instead of many sensors at the bottom reduces the numbers greatly. The right level of where to put sensors in the hierarchy would need to be considered for every scenario, depending of the requirements of the accuracy needed for sensor readings. An example of why is temperature. Obviously the temperature could be quite different inside a large container, from one end to the other.

This way, instead of considering every ware when information is needed, the appropriate parent in the hierarchy can be used.

13.4.1 AUTOMATIC AGGREGATION OF GOODS WITH SENSORS IN A HIERARCHY (SC 4.1)

When goods are transported in a hierarchy, these relationships between the goods are not permanent. They may exist for a long time, but not for ever. At a manufacturer the goods would be packed in for example boxes and then loaded on pallets. To scan every ware, in each box, to build a hierarchy would take a lot of time done by hand. Therefore by setting up gates reading tags of every ware, creating the hierarchy and the new box, this could be done automatically. The same can then be done on pallets, and so on upwards in a hierarchy. The other way around, a gate at the arrival destination could do the opposite. The gate is then used to remove the hierarchy relationships between the goods, before they are opened.

13.5 PROXIMITY CONTROL (SC 5)

When goods are transported they are often or always stacked and stored beside each other. Enabling sensors to detect awareness and position of other sensors will enable the knowledge of relative position, mentioned briefly in SC 2. Each ware also has a specification of content. This combined with relative position could help avoid unwanted close proximity. There are two solutions; it could be done either by mapping relative positions manually, or automatically. If sensors can map relative positions automatically, the knowledge of each sensor could be extended to include a list of EPC-classes not wanted in different proximities. In this way, a sensor could notify of dangerous goods placed too close. Another possibility of automatical control is to just enable the sensors to detect relative positions and communicate these to readers. Then a capturing application, or an accessing application, could use the information to either alert or influence different calculations.

13.5.1 FRESH FOOD (SC 5.1)

The quality deterioration, as mentioned in SC 1, is influenced mainly by temperature. Another important issue is how some fresh food reacts to other similar types of fresh food, e.g. apples and bananas. Storing apples in close proximity to bananas increases the deterioration rate drastically for the bananas. Detection and notification of such a case is important. Either to move goods to avoid the deterioration or to consider when calculating expiration dates. The information about these proximities could also in other cases be used to hasten ripening of fruit or other types of food.

13.5.2 DANGEROUS GOODS (SC 5.2)

When transporting chemicals, the proximity control becomes more important. If a transportation company is transporting goods from several producers, the safety could be insured by proximity control. This could be done by making each ware capable of detecting whether dangerous goods are too close. Control of the amount of a type of goods in a given space can also be interesting. There are e.g. rules for how much explosives can be stored in the same container.

There is usually no reason to know afterwards if two dangerous types of goods have been stored together, because of the obvious risk of explosion or toxic reactions. Therefore the sensors need to automatically detect proximity and recognize types of dangerous goods. In this way an alarm or notification would trigger instantly when two dangerous types of goods come too close.

14 REQUIREMENTS SPECIFICATION

This chapter will define and explain the requirements deciding how the system should be developed. The requirements are based on the scenarios previously described in Chapter 13.

First of all this chapter gives a scope for the system, because of the time limit and separation of concerns this thesis can't solve everything. Then a list of requirements are stated and explained in details, before they are mapped to scenarios and research questions. This is done to show how each requirement is needed by different scenarios, and how this thesis will answer its research questions by focusing on these requirements. After the functional requirements are stated, non-functional requirements are presented as quality attributes. This chapter will be used extensively later when creating a software architecture for this system.

14.1 SCOPE

Sensors and RFID tags are the basis of the system solution, but this thesis' focus will be above this layer. Either sensors communicate readings automatically through portals and readers, or they can be read manually. Sensors could even communicate between themselves to limit the need for central communication infrastructures.

The system solution will focus on historical data and retrieval of these for business processes or receiving applications, both inside the enterprise and outside. The need for a standardization of sharing of data across enterprises will be heavily explored.

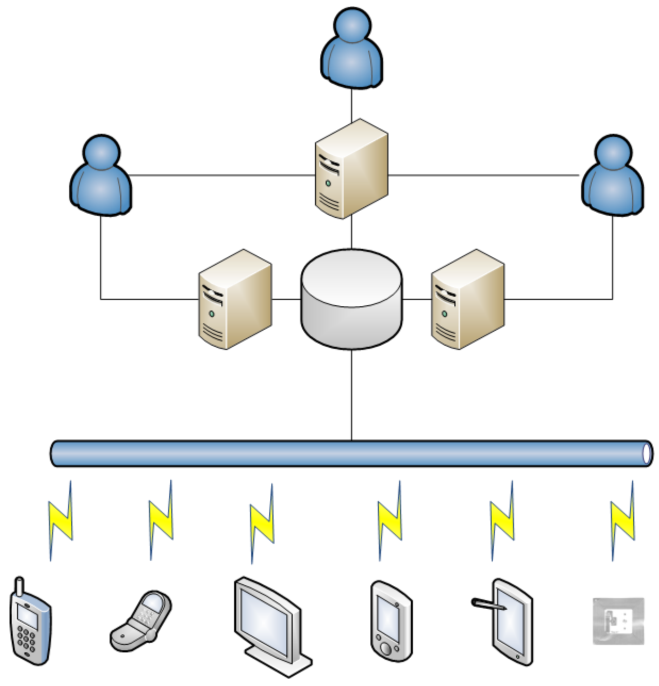


Figure 16 Overview of the environment

Figure 16 illustrates the context for this thesis. At the bottom there are tags and sensors, and data from them are pushed through a communication link into a repository. These data could be generated by using cell phones, computers, PDA or tablets, or they could be generated by sensors and tags in contact with readers. At the top, users can use different applications, called accessing applications, to retrieve this information. One example of an accessing application could be the Norwegian company Posten and their package tracking by ID on the Internet (Posten 2008). Posten is a Norwegian postage, logistics and transportation office. Other accessing applications could be more complex systems like for example a data warehouse solution for an external enterprise, accessing information about relevant products.

14.2 FUNCTIONAL REQUIREMENTS

This section describes the functional requirements required by the system to enable implementation of every scenario described in Chapter 13, and to help answer the research questions of this thesis. First the requirements are presented in an ordered list, and then every requirement is described in detail.

FR 1. The system must support data from different sensor types
  a. It must support configuration of new sensor types from a generic sensor type

FR 2. The system must be able to receive real-time data from readers
  a. Sensor data combined with a time aspect
  b. Identification of items

FR 3. The system must extend a logistic information system based on identification with a context-aware information system.

FR 4. The system must be integrated with EPCIS
  a. Generate EPCIS Events
  b. Submit EPCIS Events through an EPCIS Capture Interface

FR 5. The system must support a hierarchical structure of items
  a. A generic model for hierarchy

FR 6. Items must have a unique identification (URI)

FR 7. The system must support zero to many sensors on items
  a. Identification of each sensor.

FR 8. The system must support relative positions of items
  a. Relative position to other items in a given space
  b. A generic model for relative position of items

FR 9. The system must have a generic alert mechanism
  a. Enable alarms and alarm handling
  b. Able to define sensor types to watch, and laws for these types
  c. Able to execute a procedure if laws are broken.

FR 10. The system must support sharing of sensor information using SensorML

FR 1 THE SYSTEM MUST SUPPORT DATA FROM DIFFERENT SENSOR TYPES

This requirement is crucial for the system solution. There are already several sensor types, and new sensors and sensor data need to be easy to integrate. Almost every scenario depends on this requirement, and realization of FR 1 would answer parts of the research questions RQ 3 and RQ 6; how to share data using SensorML with EPCIS (RQ 6) and how to build a generic architecture for sensors and tags (RQ 3).

FR 2 THE SYSTEM MUST BE ABLE TO RECEIVE REAL-TIME DATA FROM READERS

Logging or storing of sensor data, historically, is needed for tracking of goods' context. The scenarios regarding temperature profiles and food deterioration are all dependent on this requirement. This will also answer parts of the research questions about extending id tracking information systems and scalability (RQ 1 and RQ 6).

FR 3 THE SYSTEM MUST EXTEND A LOGISTIC INFORMATION SYSTEM BASED ON IDENTIFICATION WITH A CONTEXT-AWARE INFORMATION SYSTEM

This requirement relates directly to RQ 1, and realization of this requirement will have a deep impact on the answer for RQ 1. It will also answer part of RQ 2, about relative positions of items. The scenarios relevant to this requirement are quality deterioration, temperature profiles, hierarchy of goods with sensors and proximity control.

FR 4 THE SYSTEM MUST BE INTEGRATED WITH EPCIS

This requirement will contribute to answering the research questions about integration with EPCIS, RQ 5, and RQ 6. The scenarios depending on this requirement will be scenarios based on sharing of information. Reloading of shipment and proving quality of fresh food both need tracking information about goods. This information could either be saved locally or retrieved from partners. Information about hierarchy and proximity also depends on EPCIS integration. This is because of aggregation events to build a hierarchy and information about products for proximity control.

FR 5 THE SYSTEM MUST SUPPORT A HIERARCHICAL STRUCTURE OF ITEMS

This requirement will help answering RQ 1, RQ 3 and RQ 4. It will tell something about how to extend an ID tracking information system with sensor information, how to build a generic architecture independently of case and how to ensure scalability. This requirement is needed by many scenarios, because items need to be placed in a hierarchy because of scalability concerns. With logistics handling several items this requirement is crucial.

FR 6 ITEMS MUST HAVE A UNIQUE IDENTIFICATION (URI)

This requirement will give information about how to extend an ID-based tracking information system with sensor data. The requirement is required for every scenario about tracking of items and their context. An identification of items is crucial if information is saved for later use, or the case is dependent on a description of items.

FR 7 THE SYSTEM MUST SUPPORT ZERO TO MANY SENSORS ON ITEMS

This requirement is crucial because every scenario is based on sensors used on items. It will answer the research question about a generic architecture, RQ 3.

FR 8 THE SYSTEM MUST SUPPORT RELATIVE POSITIONS OF ITEMS

Relative positions of items are used in the scenarios regarding temperature profiles and proximity control, and this requirement is needed for them to be realized. Realization of this requirement will partly answer RQ 2 about relative positioning and RQ 3 about how to build architecture for sensors independent of scenarios.

FR 9 THE SYSTEM MUST HAVE A GENERIC ALERT MECHANISM

This requirement is needed for the scenarios alerting systems or people about unwanted behavior. Reloading of shipment, sensor collaboration and intelligent goods, and proximity control are scenarios depending on this requirement. Supporting alerts through realization of the dependent scenarios will help answer the research question about a generic architecture (RQ 3).

FR 10 THE SYSTEM MUST SUPPORT SHARING OF SENSOR INFORMATION USING SENSORML

Every scenario storing information about sensor readings are dependent on this requirement, because SensorML has been chosen to store and share sensor information. This would include the scenarios about reloading of shipment, proving the quality of fresh food, the scenarios regarding temperature profiles, and lastly the proximity control regarding fresh food. Realization of the requirement, and enabling implementation of the scenarios mentioned above, will answer the research questions about SensorML integration with EPCIS (RQ 6). It will also help answer how to extend an ID-tracking information system with sensor information, and how to build a generic architecture towards sensors (RQ 1 and RQ 3).

14.3 REQUIREMENTS RELATION TO SCENARIOS AND RESEARCH QUESTIONS

This section maps each functional requirement to scenarios and research questions in the tables below. Both tables give an informative connection of the information stated in the previous section.

Table 6 Functional requirement to research question

	RQ 1	RQ 2	RQ 3	RQ 4	RQ 5	RQ 6
FR 1			X			X
FR 2	X			X		
FR 3	X	X				
FR 4					X	X
FR 5	X		X	X		
FR 6	X					
FR 7			X			
FR 8		X	X			
FR 9			X			
FR 10	X		X			X

Table 6 explains how every functional requirement (FR) is connected to the research questions (RQ). Implementation of the FRs would help answer the marked RQs.

Table 7 Functional requirement to scenarios

	SC 1.1	SC 1.2	SC 1.3	SC 2.1	SC 2.2	SC 2.3	SC 3.1	SC 3.2	SC 3.3	SC 4.1	SC 5.1	SC 5.2
FR 1	X	X	X	X	X	X	X	X	X		X	X
FR 2	X	X	X	X	X	X						
FR 3		X	X	X	X	X				X	X	X
FR 4		X	X							X	X	X
FR 5		X	X	X	X	X				X		
FR 6		X	X	X	X	X				X	X	X
FR 7	X	X	X	X	X	X	X	X	X	X	X	X
FR 8				X	X	X					X	X
FR 9		X					X	X	X		X	X
FR 10		X	X	X	X	X					X	

Table 7 explains how every functional requirement (FR) is connected to the scenarios (SC). Implementation of the FRs would enable the marked scenarios to be realized.

14.4 QUALITY ATTRIBUTES

This section defines the different quality attributes and selects the most important of these for this system. The quality attributes describe non-functional requirements. They are stated to stress their importance when building the system, and will later result in quality tactics in section 15.2.2.4. After the most important attributes are selected, they will each be explained in detail.

The quality attributes considered are stated in (Bass, Clements et al. 2005): Availability, modifiability, performance, security, testability, usability, scalability and portability. This system depends on all of these, but some are more important than others. Security, scalability, modifiability and performance would be the most crucial aspects of the system, and privacy will also be included because of its importance to consumers.

14.4.1 SECURITY

The need for unaltered and correct information is really important, because of laws imposed by governments. Another important issue is to have control over the information, both when stored and transferred between different sources or applications.

Information in the wrong hands could enable theft, or increase misuse of information. An example could be credit card thefts; they could easily be done because of knowledge about where the envelope sent from the bank is. Other dangerous scenarios would be changing descriptions of goods or produce fake goods; this will also undermine the credibility of the system.

14.4.2 PRIVACY

Privacy is important because people are skeptical. The information stored about products' movements and locations could potentially be quite sensitive. It is therefore important to focus on privacy.

Many examples could be given; one would be shops knowing exactly where a customer has bought his clothes. Because every garment is tagged with an RFID, and information about sales is stored and shared. This may not seem as a threat, but this could be used to do aggressive marketing by the stores. Thieves could even use this information to decide whoever they want to rob, what houses to visit.

Consumers will not be happy if everybody can check whatever they buy, at what time and amount. Information about people and their habits would be easy to track, and the need for a protection of privacy is important. Of course information like buying habits are useful for shops and even the government, but people should be skeptical because one side wants what the other does not.

14.4.3 SCALABILITY

Scalability is important because this domain will grow increasingly over time. If every item now marked with a barcode is to be marked with an EPC, the system has to scale. Even if only a small part of the items would be marked with EPCs in the beginning, the amount would increase drastically as the shift is made over time. The amount of sensors will also probably increase in the future. Because goods might start with sensors tracking only temperature to calculate quality, but later on would need information about humidity, light and other relevant information. The amount of goods tracked by sensors will most likely also increase over time.

14.4.4 MODIFIABILITY

Modifiability is important because technology is changing and expected future scenarios might not be correct. Therefore the system must be modifiable, to ease development and maintenance of the system over time.

New types of sensors and new types of identification methods are developed. The technology used by system applications are changing, communication protocols are changing. The technology is changing over time, and the system should embrace this technological advancement.

14.4.5 PERFORMANCE

Tracking and sensor information will flow from readers and sensors into the system at a great load. Several applications would use this system to retrieve information stored about sensors and sensor readings. The amount of applications would vary, but most likely it would be a large number because of both external and internal use of this information. It is therefore crucial to have high performance to handle the potential large load from both application retrievals and readers transmitting sensor readings.

15 ARCHITECTURAL SOLUTION

This chapter will present a software architecture combining an RFID tracking system with context information retrieved from sensors.

The chapter will first give an explanation of the architectural drivers, defining important issues influencing the software architecture. Then a section explaining different viewpoints will be presented, defining how the software architecture is best presented to its stakeholders. After the viewpoints are selected, the following section will describe and select relevant quality tactics to realize the chosen quality attributes in section 14.4Quality Attributes. The next section will describe well-known patterns, defining best practices suited for this system. Before the last section presents the software architecture, explained by using the different viewpoints selected to show several aspects of the software architecture to the system's stakeholders.

15.1 ARCHITECTURAL DRIVERS

First of all this software architecture is made to prove a connection and integration of context information with a tracking system working with IDs. This software architecture will use best practices wherever possible, but most of the issues have not been tried out yet or even solved. Wherever solutions already exist, the software architecture will focus on integration of such parts with the rest. The limited amount of time available to build this architecture would probably affect its quality.

15.2 CHOOSING VIEWPOINTS

This section will first describe what viewpoints are and why they have to be chosen. Then the different viewpoints are evaluated with regards to the stakeholders. This section will only present evaluations of the relevant viewpoints. The selected viewpoints will later be used to present the architecture in section 15.5.

15.2.1 WHAT ARE VIEWPOINTS AND WHY INVESTIGATE THEM

A viewpoint can be explained as a way of seeing something, and different people have different viewpoints. A view is what you see from the chosen viewpoint. An example inspired by (Bass, Clements et al. 2005) is building architecture. In building there is not just one sketch. For the builders, e.g. the industry workers, there exist detailed blueprints. For the buyers, normal citizens, there exist 3D models. There are also plumbing diagrams and much more. Not one of them is the architecture, but they all convey the architecture. Understanding the stakeholders and what they need is critical for the conveyance of the architecture.

In "4+1" View Model of Software Architecture (Kruchten 1995), 5 viewpoints are presented and explained. The 5 viewpoints are shown in Figure 17.

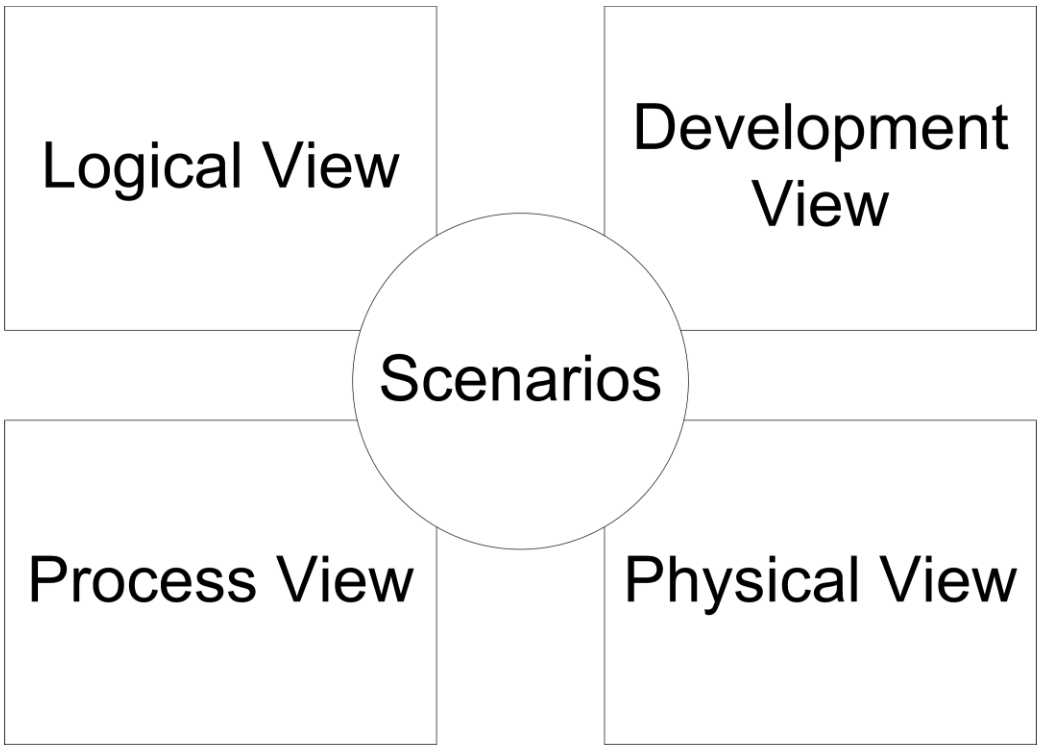


Figure 17 The 4+1 View Model (Kruchten 1995)

The logical viewpoint primarily supports the functional requirements, what the system is supposed to offer to the users. It is class diagram, consisting of objects and object classes. Typical notation for this view could be UML Class Diagrams.

The development viewpoint focuses on actual modules of the software architecture. Explaining how they interact and their responsibility. It is often used to ease allocation of work to different teams. Typical notation for this view could be UML Component Diagrams.

The process viewpoint supports both functional and non-functional requirements, displaying business processes or activities at different abstraction levels. Typical notations for this view could be UML Activity Diagrams, Business Process Modeling Notation, UML Sequence Diagrams or Data Flow Diagrams.

The physical viewpoint supports the non-functional requirements and presents the physical world. The system is executed on nodes, connected by a network. Backup nodes may exist as well as test nodes. All this can be explained and presented in the physical view.

The scenarios are referred to as the "+1", because they are redundant to the other viewpoints. The scenarios bind the different views together, explaining by examples how the system will work. As explained in (Kruchten 1995), the scenarios have two purposes:

- As a driver to discover the architectural elements during the architecture design as we will describe later
- As a validation and illustration role after this architecture design is complete, both on paper and as the starting point for the tests of an architectural prototype.

More information about the different views and the 4+1 View Model can be found in (Kruchten 1995).

15.2.2 EVALUATION OF THE DIFFERENT VIEWPOINTS

This section will evaluate the relevant viewpoints with regards to the stakeholders. The viewpoints presented are: Logical, Process, and Development. Scenarios will also be used and evaluated. For each viewpoint a stakeholder evaluation, a short explanation, and the chosen notation is given.

15.2.2.1 LOGICAL VIEWPOINT

This viewpoint shows class diagrams for parts in the system, and how they interact. The notation used for this view is UML Class diagram (Fowler 2005).

STAKEHOLDER EVALUATION

- **Transportation companies and Retailers**
  Interested in the system design, for own implementation and understanding of the system, and to be able to follow the implementation from an early state.
- **Developers**
  Developers are interested in a detailed description of the different parts of the system, for guidance during implementation. To be used to control the workflow and improve efficiency.

15.2.2.2 PROCESS VIEWPOINT

This viewpoint shows data flow diagrams (DFD) explaining information flows between external users or systems and processes, and activity diagrams describing the overall flow of control. The notations used for this view are Data Flow Diagrams (DFD) (Hawryszkiewycz 2001) and UML Activity Diagrams (Fowler 2005).

STAKEHOLDER EVALUATION

- **Government**
  Interested in how information will flow, to get an insight into new possibilities. To form new laws and regulations for sharing of important information, like tracing and quality.
- **Consumers**

Interested in what retailers will be able to know and how they can improve to better suit the consumers' needs. Interested to see new possibilities and ensure safety and privacy.

- **Product Manufacturers**
  Interested in information about the importance of new possibilities and how information is shared, and to what extent it improves today's solutions. Product manufacturers want to estimate the business value of using EPC from the very start when products are made.
- **Transportation companies**
  Interested in how information is shared with external companies, and what benefits will come from this sharing. Interested in how much of the system they need to implement to be a supplier of needed information.
- **Retailers**
  Interested in how documentation of treatment and other information flows and how it can be shared with external enterprises. Interested in new possibilities and how they can improve business processes.
- **Developers**
  Interested in how the different parts of the system interact, so work can be delegated with as few problems with integration as possible.

15.2.2.3 DEVELOPMENT VIEWPOINT

This viewpoint gives an overview over system components, showing different roles and how they are interacting. The notation used by this view is UML Component diagram (Fowler 2005).

STAKEHOLDER EVALUATION

- **Government, Product Manufactures, Transportation companies and Retailers**
  Interested in an overview, showing different parts of the system and how they interact, and to easily get an understanding of the whole system without worrying about the underlying details.
- **Developers**
  Their interests lie in the same as the rest mentioned above, but they are also interested in the underlying details. The development view will provide a solution of how to separate concerns into different parts, implementing them one by one. This will ease the delegation of work, and integration of all the parts later on.

15.2.2.4 SCENARIOS

The scenarios are used to bind the views together. This view will use two scenarios from section 13, and show how they are proposed solved by the software architecture.

STAKEHOLDER EVALUATION

All stakeholders would be interested to have an abstract explanation of the system, through real scenarios they can relate to. This will further the understanding of the concept of the software architecture.

15.3 QUALITY TACTICS

In Section 14.4, quality attributes selected as most important for the system are presented. In this section, quality tactics to be used to realize them are explained, inspired by (Bass, Clements et al. 2005). These tactics are to be used when creating the software architecture, and later when building the system.

15.3.1 MODIFIABILITY

It is important that the system is modifiable as described in Section 14.4.4. To solve this problem these tactics will be used:

Have a focus on localizing modification because there will be changes to this system. For example, new sensors will need to be added or edited, and it is hard to predict future needs. Therefore these tactics will be used to localize modification:

- Maintain semantic coherence
    - Put common services in modules, making it easy to modify one module alone without worrying about the others.
- Anticipate expected change
    - Investigate as best as possible how things will change in the future, and bring this into the solution.
- Generalize the module
    - Generalize a module to the extent that many different cases are allowed in one implementation, without the need for modification of the module.

To further improve modifiability, the solution should try to prevent ripple effects by hiding information on modules. Using interfaces to define what information is public, and hide the rest.

The last tactic is to defer binding time by using configuration files. More precisely, advanced files describing different aspects of the system, like sensors. This would enable new sensors to be added without coding and compiling new code or even rebooting the system.

15.3.2 PERFORMANCE

Performance is important due to the amount of data that will pass through the system. The number of clients and the possibly enormous amount of sensors will in general greatly affect performance. To ensure performance the following tactics will be used:

- Reduce computational overhead
  - Reducing the use of unneeded resources. Making the need for communication to a minimum if it is not required.
- Bound execution times and queue sizes
  - Limit the time and sizes to the extent that information needed is retrieved, but oversized method calls are ignored. For example by limiting the maximum time span of a query for sensor information from years to weeks or even hours.
- Maintain multiple copies of either data or computations
  - Allowing the use of several resources to be queried for information.

15.3.3 SECURITY

Security is very important for this type of system, and without it enterprises and consumers would be very skeptical. The main focus for security tactics would thus be to resist attacks, described below:

- Authentication
  - It is important to know who the retriever is, to be sure of what to do.
- Authorization
  - Different users need different information, and some users might not be granted access to specific parts of information. Having authentication, authorization to allow different users different type of access can be enabled.
- Maintaining data confidentiality
  - Ensuring correctness of information is important. To prevent the information of being tampered with before reaching the user.

15.3.4 PRIVACY

Privacy is crucial to a tracking system because of user skepticism of revealing personal information. It can be accepted as a known truth that users are skeptical to new technology. Tracking being a highly discussed privacy issue. It is therefore crucial to uphold privacy, and to do this the focus will be on the same tactics stated in the previous section about security.

15.3.5 SCALABILITY

Because of the vast amount of potential sensors recording measurements and clients retrieving information, scalability is important for the system. A focus will be put on the tactics mentioned in the section about performance, and also the reuse of systems built for scalability. It is important to perform as much computation as possible in the edge of the system, preventing the need to pass everything through a central server.

15.4 PATTERNS

This section will describe well-known patterns, defining best practices, suited for this system. These are used later when creating the software architecture. Every pattern is explained before an explanation is given of why they are used.

15.4.1 SEPARATION OF CONCERNS

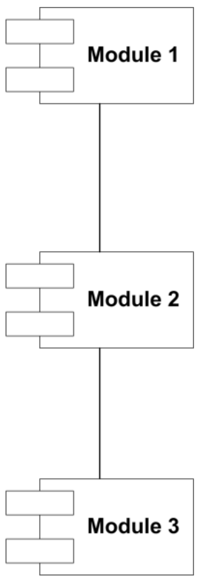


Figure 18 Separation of concerns

By grouping or dividing concerns into different modules they can be solved separately. The groups can be seen as members with high cohesion, solving their domain without worrying about everything else. This way all groups would not solve everything, and duplicate work could be avoided. By allowing others to solve one problem, the rest would have one problem less and could solve their problems better. The result would be better solutions for every module, because the concerns would only be for the specific module.

Separation of concerns will be used because the system is focused on bringing two domains together, tracking goods based on identification and context-information tracked by sensors. The concerns are split into separate modules, for example one sensor module and one ID tracking module. The concerns are then solved separately, using best practices for the separate parts or modules greatly improving the system.

15.4.2 FACADE

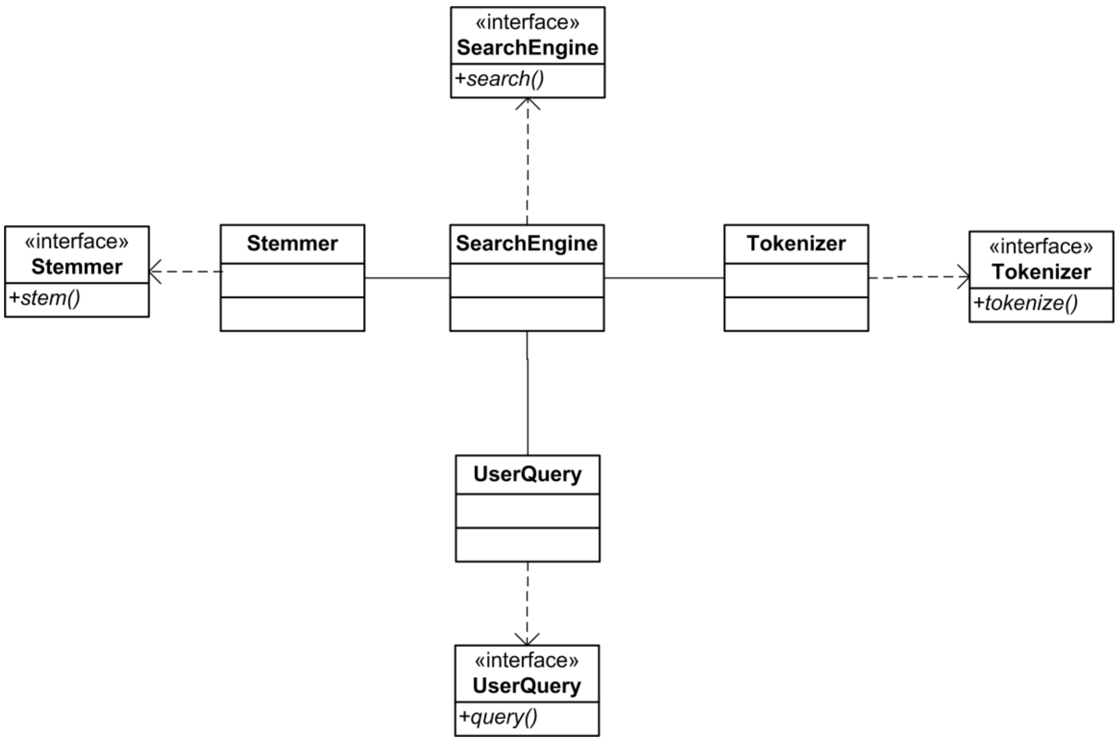


Figure 19 Facade pattern

The facade pattern as it is explained in (Gamma 1994), is about creating abstract interfaces for complex operations. The goal is to make it easier to use for external parts. In Figure 19, the overall concept of a search engine is explained. Whenever a user types a query (represented as the *search()* method in *SearchEngine*), different jobs have to be done before the results can be returned. First the *tokenize()* method must be called, then the *stem()* method, and finally the *query()* method. Instead of the systems using every method one by one before a search, it only worries about *SearchEngine* and the method *search().* The result being the same as if every method was called separately, but instead of using many methods only one method implemented by *SearchEngine* is used.

The facade pattern will be used to supply only a few interfaces, and have the underlying system take care of the rest. These few interfaces will integrate the functionality from the underlying systems, serving the needs of external applications.

15.5 VIEWS

This section will present each view resulting from the selection of viewpoints in section 15.2. A view is what can be seen from the chosen viewpoint, showing different aspects or details of the same object. Every view will therefore describe different aspects of the software architecture. Describing in detail a proposed solution for the system to the stakeholders, in a way they all can understand.

The views are presented as figures with text explaining different parts and aspects of the view. The views are presented in the following order: logical view, process view and the development view. At last the scenarios will bind the three views together to further explain the software architecture and give a greater understanding for all stakeholders.

15.5.1 LOGICAL VIEW

This view will first present a class diagram showing every element of the logical view, before different groups are presented to give a better presentation. Every group will describe different elements having high cohesion. The groups are EPCIS, Sensor Web Enablement and Context-Aware System. All the elements in every group will be described in detail.

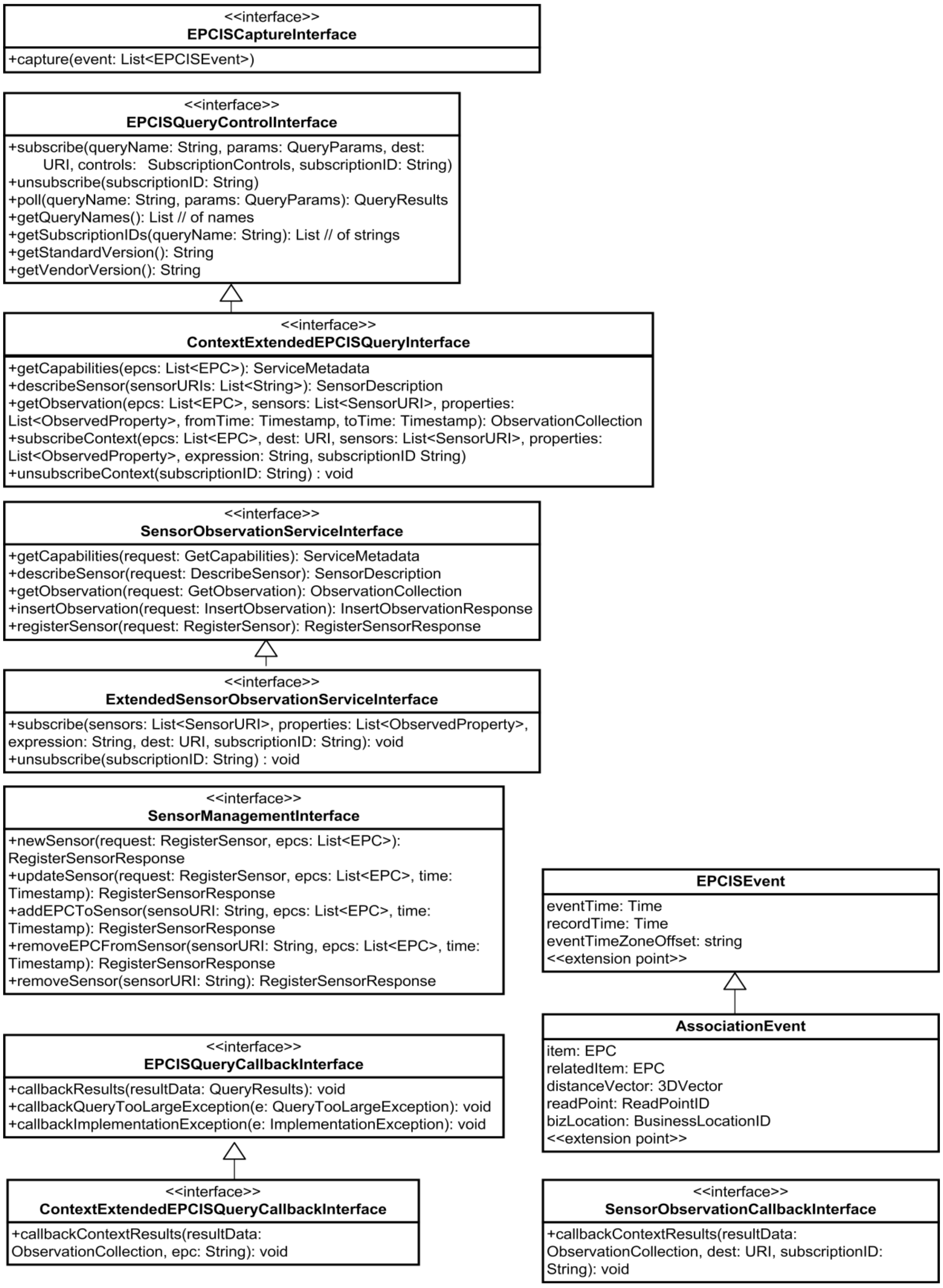
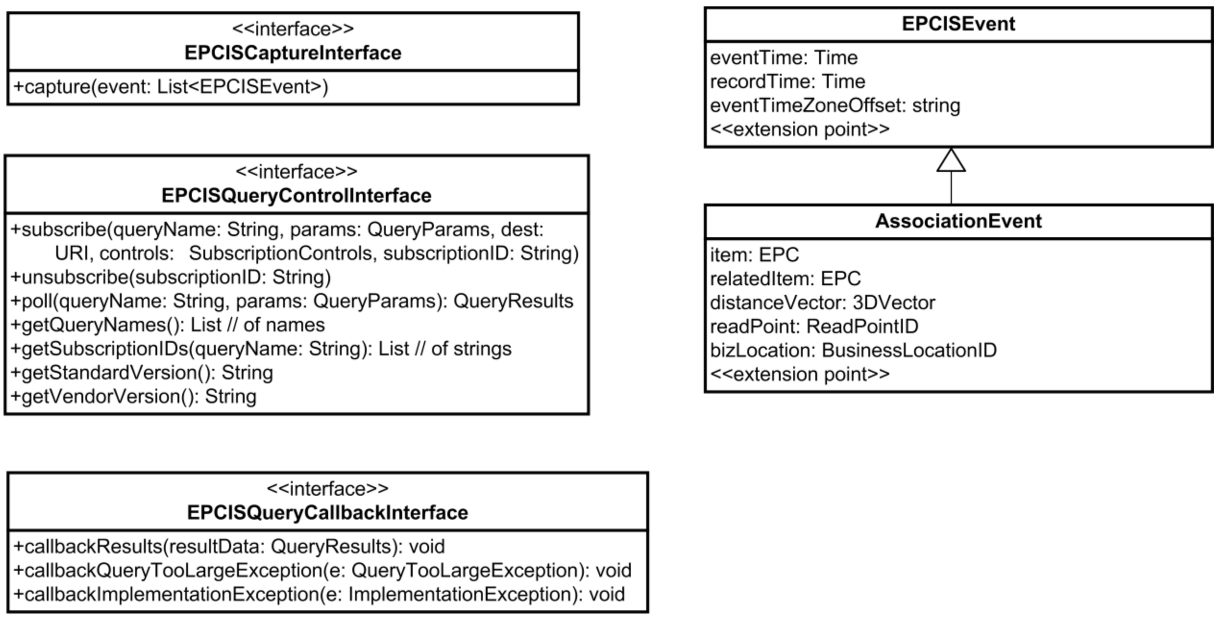

<<interface>>
EPCISCaptureInterface
+capture(event: List<EPCISEvent>)
<<interface>>
EPCISQueryControlInterface
+subscribe(queryName: String, params: QueryParams, dest: URI, controls: SubscriptionControls, subscriptionID: String)
+unsubscribe(subscriptionID: String)
+poll(queryName: String, params: QueryParams): QueryResults
+getQueryNames(): List // of names
+getSubscriptionIDs(queryName: String): List // of strings
+getStandardVersion(): String
+getVendorVersion(): String
<<interface>>
ContextExtendedEPCISQueryInterface
+getCapabilities(epcs: List<EPC>): ServiceMetadata
+describeSensor(sensorURIs: List<String>): SensorDescription
+getObservation(epcs: List<EPC>, sensors: List<SensorURI>, properties: List<ObservedProperty>, fromTime: Timestamp, toTime: Timestamp): ObservationCollection
+subscribeContext(epcs: List<EPC>, dest: URI, sensors: List<SensorURI>, properties: List<ObservedProperty>, expression: String, subscriptionID String)
+unsubscribeContext(subscriptionID: String) : void
<<interface>>
SensorObservationServiceInterface
+getCapabilities(request: GetCapabilities): ServiceMetadata
+describeSensor(request: DescribeSensor): SensorDescription
+getObservation(request: GetObservation): ObservationCollection
+insertObservation(request: InsertObservation): InsertObservationResponse
+registerSensor(request: RegisterSensor): RegisterSensorResponse
<<interface>>
ExtendedSensorObservationServiceInterface
+subscribe(sensors: List<SensorURI>, properties: List<ObservedProperty>, expression: String, dest: URI, subscriptionID: String): void
+unsubscribe(subscriptionID: String) : void
<<interface>>
SensorManagementInterface
+newSensor(request: RegisterSensor, epcs: List<EPC>): RegisterSensorResponse
+updateSensor(request: RegisterSensor, epcs: List<EPC>, time: Timestamp): RegisterSensorResponse
+addEPCToSensor(sensoURI: String, epcs: List<EPC>, time: Timestamp): RegisterSensorResponse
+removeEPCFromSensor(sensorURI: String, epcs: List<EPC>, time: Timestamp): RegisterSensorResponse
+removeSensor(sensorURI: String): RegisterSensorResponse
EPCISEvent
eventTime: Time
recordTime: Time
eventTimeZoneOffset: string
<<extension point>>
AssociationEvent
item: EPC
relatedItem: EPC
distanceVector: 3DVector
readPoint: ReadPointID
bizLocation: BusinessLocationID
<<extension point>>
<<interface>>
EPCISQueryCallbackInterface
+callbackResults(resultData: QueryResults): void
+callbackQueryTooLargeException(e: QueryTooLargeException): void
+callbackImplementationException(e: ImplementationException): void
<<interface>>
ContextExtendedEPCISQueryCallbackInterface
+callbackContextResults(resultData: ObservationCollection, epc: String): void
<<interface>>
SensorObservationCallbackInterface
+callbackContextResults(resultData: ObservationCollection, dest: URI, subscriptionID: String): void

Figure 20 Logical view

Figure 20 presents the logical view of the software architecture solution, and because of the number of elements this figure will be presented in separate parts.

15.5.1.1 EPC INFORMATION SERVICES

This section describes the group related to EPCIS. All elements except the *AssociationEvent* are from EPCIS. The group is presented in Figure 21 and each element is then described in detail. More information about the elements from EPCIS can be found in Chapter 7 and Appendix B.



<<interface>>
EPCISCaptureInterface
+capture(event: List<EPCISEvent>)
<<interface>>
EPCISQueryControlInterface
+subscribe(queryName: String, params: QueryParams, dest: URI, controls: SubscriptionControls, subscriptionID: String)
+unsubscribe(subscriptionID: String)
+poll(queryName: String, params: QueryParams): QueryResults
+getQueryNames(): List // of names
+getSubscriptionIDs(queryName: String): List // of strings
+getStandardVersion(): String
+getVendorVersion(): String
<<interface>>
EPCISQueryCallbackInterface
+callbackResults(resultData: QueryResults): void
+callbackQueryTooLargeException(e: QueryTooLargeException): void
+callbackImplementationException(e: ImplementationException): void
EPCISEvent
eventTime: Time
recordTime: Time
eventTimeZoneOffset: string
<<extension point>>
AssociationEvent
item: EPC
relatedItem: EPC
distanceVector: 3DVector
readPoint: ReadPointID
bizLocation: BusinessLocationID
<<extension point>>

Figure 21 EPCIS parts of the system

EPCIS CAPTURE INTERFACE

This interface has one method, receiving EPCIS Events. This interface is defined by EPCglobal. The interface defines the communication between the Capturing Application and EPCIS Repository.

EPCIS QUERY CONTROLLER INTERFACE

This interface is often referred to as EPCIS Query Interface, and it is defined by EPCglobal. The purpose of this interface is to give Accessing Applications the ability to receive EPCIS Events. Whether they pull for information or subscribe and EPCIS pushes information to them. The queries used are predefined and therefore referred to by name, used by both *subscribe()* and *poll().* In the *subscribe()* method there is also a parameter for an URI to where EPCIS Events should be sent.

EPCIS QUERY CALLBACK INTERFACE

This interface is for receiving query results when subscribing through the *EPCIS Query Controller Interface*. The subscriber needs to implement this interface to be able to receive answers from the EPCIS Repository, when subscribing for EPCIS Events. There are two exception methods in this interface to be able to communicate exceptions, either *QueryTooLargeException* or *ImplementationException*.

EPCIS EVENT

This is the basic EPCIS Event defined by the EPCIS Architecture Framework. This event is only a generic event, and other events such as *ObjectEvent*, *AggregationEvent*, *QuantityEvent* and *TransactionEvent* are subclasses of this. The event is just stating the time of recording and the time of occurrence.

ASSOCIATION EVENT

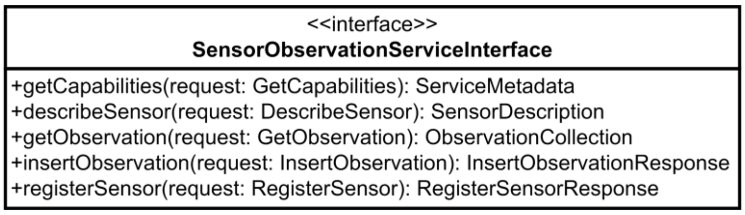
This event extends the *EPCISEvent* and is used to associate one EPC with another EPC. The *AssociationEvent* is the only element in this section proposed by the thesis as an extension of EPCIS. Different from the *EPCISAggregationEvent (see Appendix B)*, describing a solid connection between EPCs, this event represent a much lighter connection. The *EPCISAggregationEvent* is designed for items packed in boxes for a long time, but this *AssociationEvent* is for two boxes or items placed next to each other. The two boxes could easily be moved from each other, without unwrapping before separating them.

The most important parameter of the *AssociationEvent* is the *DistanceVector* telling how the two EPCs are related, the *item* and *relatedItem*. This is a vector describing distance and direction, with the *item*'s position as point of origin.

The *AssociationEvent* does not include many of the normal EPCIS Event properties, such as *bizStep* and *disposition* included in *EPCISAggregationEvent* and the rest of the events from EPCIS. This is because the *AssociationEvent* would be used as an additional event, and it will only be available for Accessing Applications with knowledge of this extension. For all the other Accessing Applications there need to be a normal EPCIS Event describing the event. A model of all the standard EPCIS Event types can be found in Appendix B.

15.5.1.2 SENSOR WEB ENABLEMENT

This section describes the group related to Sensor Web Enablement (SWE). The group is presented in Figure 22 and each element is then described in detail. More information about the elements from SWE can be found in Chapter 9.



<<interface>>
SensorObservationServiceInterface
+getCapabilities(request: GetCapabilities): ServiceMetadata
+describeSensor(request: DescribeSensor): SensorDescription
+getObservation(request: GetObservation): ObservationCollection
+insertObservation(request: InsertObservation): InsertObservationResponse
+registerSensor(request: RegisterSensor): RegisterSensorResponse

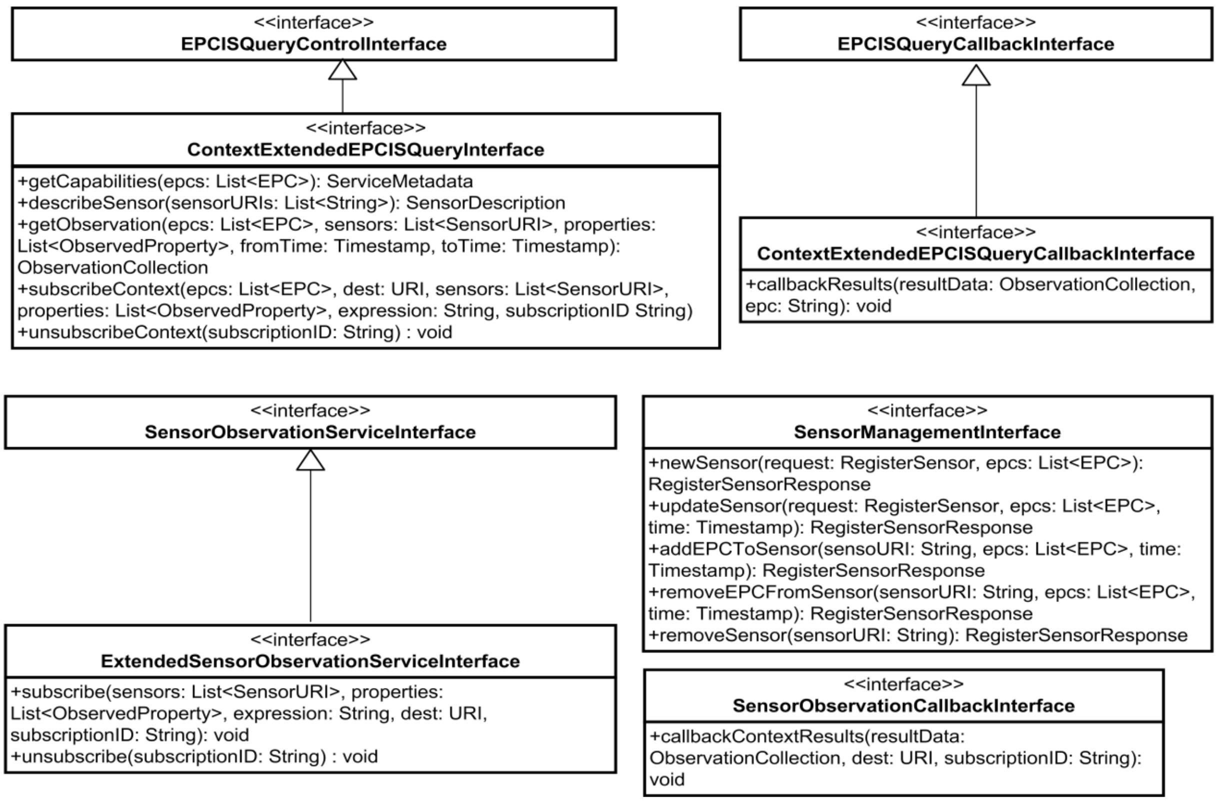
Figure 22 Sensor Web Enablement parts of the system

SENSOR OBSERVATION SERVICE INTERFACE

This interface is from the SWE framework. The interface is extended in the Context-Aware Goods System, and more information about this interface can be found in Chapter 9.

15.5.1.3 CONTEXT-AWARE GOODS SYSTEM

This section describes the group representing the Context-Aware Goods System (CAGS). The group is presented in Figure 23 and each element is then described in detail.



<<interface>>
EPCISQueryControlInterface
<<interface>>
EPCISQueryCallbackInterface
<<interface>>
ContextExtendedEPCISQueryInterface
+getCapabilities(epcs: List<EPC>): ServiceMetadata
+describeSensor(sensorURIs: List<String>): SensorDescription
+getObservation(epcs: List<EPC>, sensors: List<SensorURI>, properties: List<ObservedProperty>, fromTime: Timestamp, toTime: Timestamp): ObservationCollection
+subscribeContext(epcs: List<EPC>, dest: URI, sensors: List<SensorURI>, properties: List<ObservedProperty>, expression: String, subscriptionID String)
+unsubscribeContext(subscriptionID: String) : void
<<interface>>
ContextExtendedEPCISQueryCallbackInterface
+callbackResults(resultData: ObservationCollection, epc: String): void
<<interface>>
SensorObservationServiceInterface
<<interface>>
SensorManagementInterface
+newSensor(request: RegisterSensor, epcs: List<EPC>): RegisterSensorResponse
+updateSensor(request: RegisterSensor, epcs: List<EPC>, time: Timestamp): RegisterSensorResponse
+addEPCToSensor(sensoURI: String, epcs: List<EPC>, time: Timestamp): RegisterSensorResponse
+removeEPCFromSensor(sensorURI: String, epcs: List<EPC>, time: Timestamp): RegisterSensorResponse
+removeSensor(sensorURI: String): RegisterSensorResponse
<<interface>>
ExtendedSensorObservationServiceInterface
+subscribe(sensors: List<SensorURI>, properties: List<ObservedProperty>, expression: String, dest: URI, subscriptionID: String): void
+unsubscribe(subscriptionID: String) : void
<<interface>>
SensorObservationCallbackInterface
+callbackContextResults(resultData: ObservationCollection, dest: URI, subscriptionID: String): void

Figure 23 Context-Aware Goods System parts

CONTEXT EXTENDED EPCIS QUERY INTERFACE

This interface extends the *EPCISQueryControllerInterfaces*, with methods for context information retrieval.

The first method *getCapabilities()* retrieves information about what types of context are tracked for the given EPCs.

The second method *describeSensor()* retrieves information about the given sensors in SensorML format, the input is sensor URI for identification of sensors. The sensor URIs can be retrieved using the *getCapabilities()* method.

The method *getObservation()* retrieves actual sensor readings i.e. temperature over time. It is possible to give a list of EPCs, interesting EPCs; a list of sensors, interesting sensors; a list of properties, e.g. temperature if a sensor has several; start time and end time as inputs. All of these inputs could be empty, and if for example the sensor list is empty information from every sensor for the given EPCs are retrieved. If both EPCs and sensor URIs are given as input, information from the given sensors with information about the given EPCs would be retrieved. The only requirement is that either a list of EPCs or a list of sensor URIs is given.

The subscription method, *subscribeContext()*, is for registering a request for information or alerts the instant it occurs. The method takes several parameters, but some can be excluded. The obligated parameters are: destination URI, properties, subscriptionID and one of the two: sensor URIs or EPCs.

The list of EPCs, properties, and SensorURIs are the same as in *getObservations().* The *subscriptionID* is a string uniquely identifying the subscription for the subscriber. The *subscriptionID* together with the identity of the accessing application is unique for CAGS. The *expression* parameter is a logical expression with mathematical operators like >, <, =, And, Or and combinations of these with values. An example could be “((temp > 0) AND (temp < 5))”, combined with the other parameters it could mean fresh food with temperatures between 0 and 5 degrees Celsius. After registering this request, observations matching the first parameters (EPCs, sensors and properties) and breaking the expression would be communicated to the subscriber.

BNF (Backus Naur Form) is used to describe syntax (Van Roy and Haridi 2004). The notation is as follows:

- ::= meaning "is defined as"
- | meaning "or"
- < > angle brackets used to surround category names.
- ‘’ are used to show a symbol, ‘(‘ would explain that the character “(“ is to be used.

The *expression* parameter is here specified in BNF:

<expression> ::= ‘(‘ <leafExpression> <operator> <leafExpression> ‘)’ | <leafExpression>

<leafExpression> ::= <expression> | ‘(‘ <digit> <comparator> <paramName> ‘)’ | ‘(‘ <paramName> <comparator> <digit> ‘)’

<operator> ::= AND | OR

<comparator> ::= ‘>’ | ‘<’ | ‘>=’ | ‘<=’ | ‘=’

A *parameterName* is parameters from the list of *properties*, meaning the *memberName* identifying the *property*. A digit is any number, decimals are allowed. A correct expression would be: “((temp > 0) AND (temp < 5))” as mentioned above.

If EPCs are not given, the list of sensor URIs is used. If the expression is empty, all observations matching other parameters are to be communicated to the subscriber.

To cancel a subscription, the *unsubscribeContext()* method is used. This method takes only one parameter, *subscriptionID*, given as input parameter when the subscription was created.

CONTEXT EXTENDED EPCIS QUERY CALLBACK INTERFACE

This interface is for receiving observation and query results when subscribing through the *ContextExtendedEPCISQueryControllerInterface*. The subscriber needs to implement this interface to be able to receive answers from the Context-Aware Goods System, when subscribing for information. If a list of EPCs is given when subscribing, the observations will be related to the given EPCs. If no EPCs are given and a list of sensor URIs was used, only observations will be received.

SENSOR MANAGEMENT INTERFACE

This interface is for managing sensors; creating, editing and deleting. It is also used for editing of the relation of EPC to sensor. Controlling which sensor is related to which EPCs, in a given time span. An example could be a warehouse with a cold storage. Whenever goods are stored in the cold storage, information about temperature can be retrieved by using the storage’s sensor. So whenever goods enter and exit, the information has to be registered through this interface. Then CAGS will know that from time X to time Y, this sensor has information about this EPC. When registering the information, a timestamp can also be given. This is done to enable the informant to send delayed information without losing observation information.

EXTENDED SENSOR OBSERVATION SERVICE INTERFACE

It is used by both the Sensor Capturing Application and the Context-Aware Goods System to manage sensor observations information, one of them inserting and the other retrieving them later on. Retrieval of sensor observations can be done in two ways, either by pulling the information from the system or by subscribing to specific observations. In much the same way explained in the *ContextExtendedEPCISQueryInterface.* This interface is also used to register and manage sensors in the Sensor Observation Service (SOS).

SENSOR OBSERVATION CALLBACK INTERFACE

This interface is needed by the SOS to send observations that CAGS subscribe for, because an Accessing Application has subscribed through the CAGS for this information. This is done because CAGS map sensor URIs to EPCs, and this interface only focuses on sensor URIs.

15.5.2 PROCESS VIEW

This view will first present a context flow diagram giving an abstract view of the data flow between the system and stakeholders, before a more detailed diagram is presented to give further insight. Then an activity diagram will show what happens when a sensor reading enters the system.

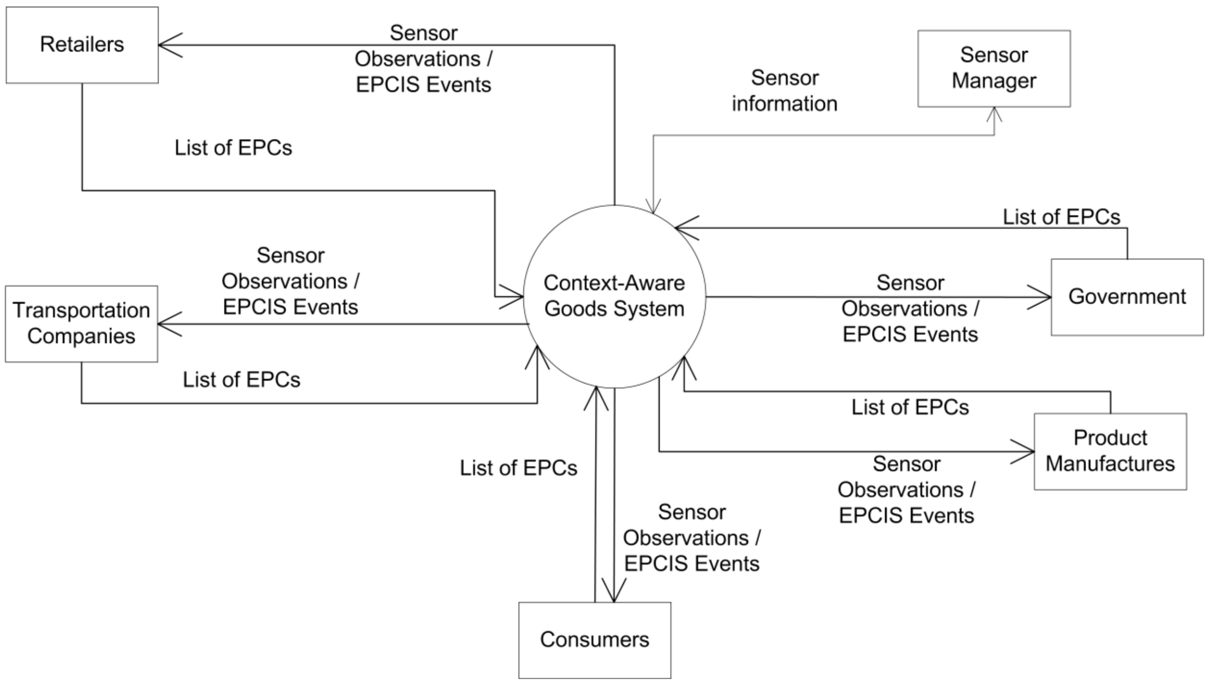

Retailers
Sensor Observations / EPCIS Events
Sensor information
Sensor Manager
List of EPCs
List of EPCs
Sensor Observations / EPCIS Events
Transportation Companies
Context-Aware Goods System
Sensor Observations / EPCIS Events
Government
List of EPCs
List of EPCs
Product Manufactures
List of EPCs
Sensor Observations / EPCIS Events
Sensor Observations / EPCIS Events
Consumers

Figure 24 Context data flow diagram

Figure 24 shows the information flow between the users, stakeholders, and the system. The focus is on what is specific for the system, and not on the underlying EPCIS and SWE systems. The information entered will mainly be lists of EPCs and sensor information, and the information sent out is sensor observations and EPCIS events.

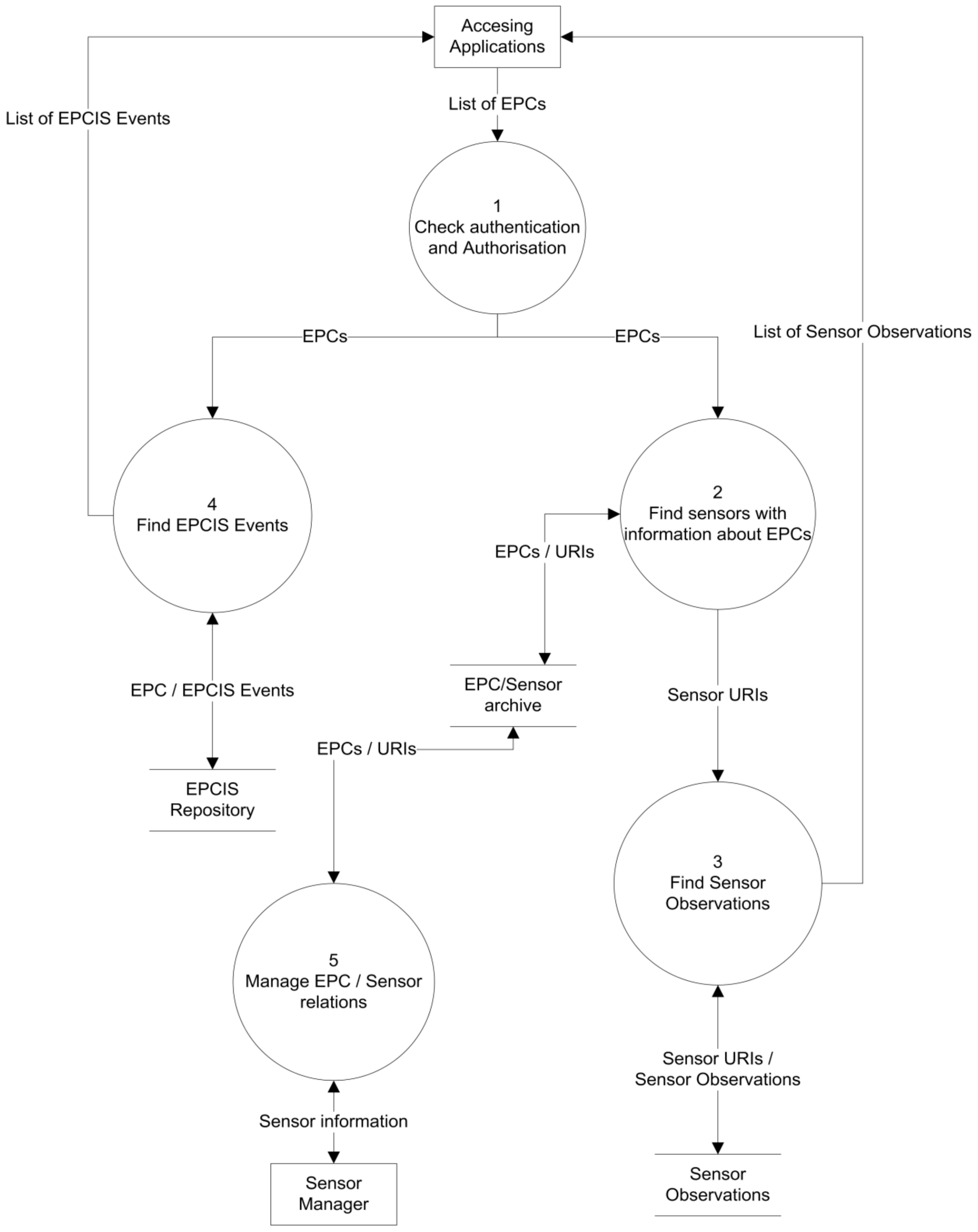


Figure 25 Top level data flow diagram

Figure 25 shows a more detailed process view of the system. The different stakeholders, or users, are now presented as Accessing Applications. Every stakeholder would access the system through an accessing application, either directly or indirectly through other systems. The last role, Sensor Manager, is a role for sensor setup and configuration. This role would be taken by many of the stakeholders, but not all. The use of the Accessing Application and the Sensor Manager actors instead of every stakeholder is done to make the diagram easier to understand.

The accessing applications are sending lists of EPCs to the system to retrieve information. The information returned are either EPCIS Events or sensor observations, based on the method invoked. The sensor manager is sending and receiving sensor information. This information is used to manage all the sensors; what they are and how they work, what items are they attached to and such administration of sensors. The actor's dataflow moves through several processes as shown in Figure 25, processes shown as circles in the figure. These will now be explained in detail.

1 CHECK AUTHENTICATION AND AUTHORIZATION

This process explains the need for security, done by authentication and authorization. It is important that no one without access to perform requests can retrieve information. This process does not say anything about how it should be done. Authorization is also explained, because different accessing applications might not have access to the same information. One could for example have access to the EPCIS Events; others could additionally have access to sensor observations.

2 FIND SENSORS WITH INFORMATION ABOUT EPC

This process uses the information about sensors related to EPCs at given time spans to select sensors based on incoming EPCs. A list of EPCs is sent to this process, and then the process maps each EPC to known sensors with information about this EPC. If the user have included a time span in the request, only sensors with information from that time span will be used. The mapping information about EPCs to sensor URIs is retrieved from an *EPC/Sensor archive*.

3 FIND SENSOR OBSERVATIONS

This process uses the incoming sensor URIs to retrieve observations from a repository of *Sensor Observations.* This is done using the Sensor Observation Service interfaces defined in SWE, described in the logical view, section 15.5.1.

4 FIND EPCIS EVENTS

This process retrieves relevant EPCIS Events to the EPCs given as input to the process. This is done as a normal EPCIS query to an *EPCIS Repository.*

5 MANAGE EPC / SENSOR RELATIONS

This process registers connections between sensors and EPCs, and manages sensor information. Whenever a new type of sensor is set up, this information is sent to this process. Additionally, whenever sensors are allocated to EPCs, e.g. by attaching a sensor to an item, this information is also sent to this process. This process updates *EPC/Sensor archive*, based on information from the Sensor Manager.

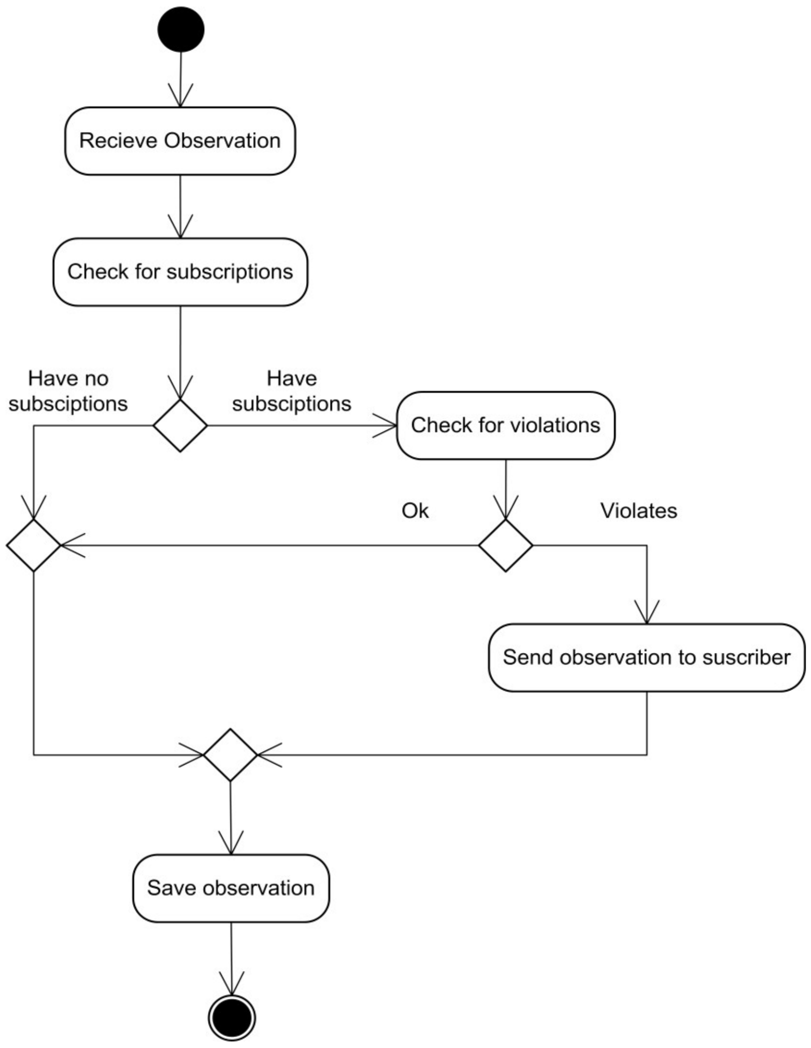


Figure 26 Observation registration

Figure 26 Observation registration shows the activity performed whenever a sensor observation is retrieved and sent to the system. This activity is presented to show how the alert mechanism works.

The observation is received as a first activity, and then the system will check for subscribers for this sensor property. If there are no subscribers the information is stored for later retrieval, but if there are subscribers they need to be investigated. Every subscriber has given rules of violation, and if the observation violates the subscriber's rule, an alert will be sent. The alert will be sent by sending the observation to the subscriber, and then the observation is saved for later retrieval. If the observation does not violate any rules defined by the subscribers, the observation is directly saved for later retrieval. An example of such a rule could be a temperature sensor, setting maximum and minimum of allowed readings. If the temperature observations move outside this boundary, the rules are violated.

15.5.3 DEVELOPMENT VIEW

This section defines roles and interfaces for communication. There is no need for every role to be implemented separately. Several roles or interfaces could be included in one application. The only real requirement is the support of the *ContextExtendedEPCISQueryInterface* by an underlying system.

The section will first present the view in Figure 27, and then the roles and interfaces are described. The roles are described in detail first, and then the interfaces.

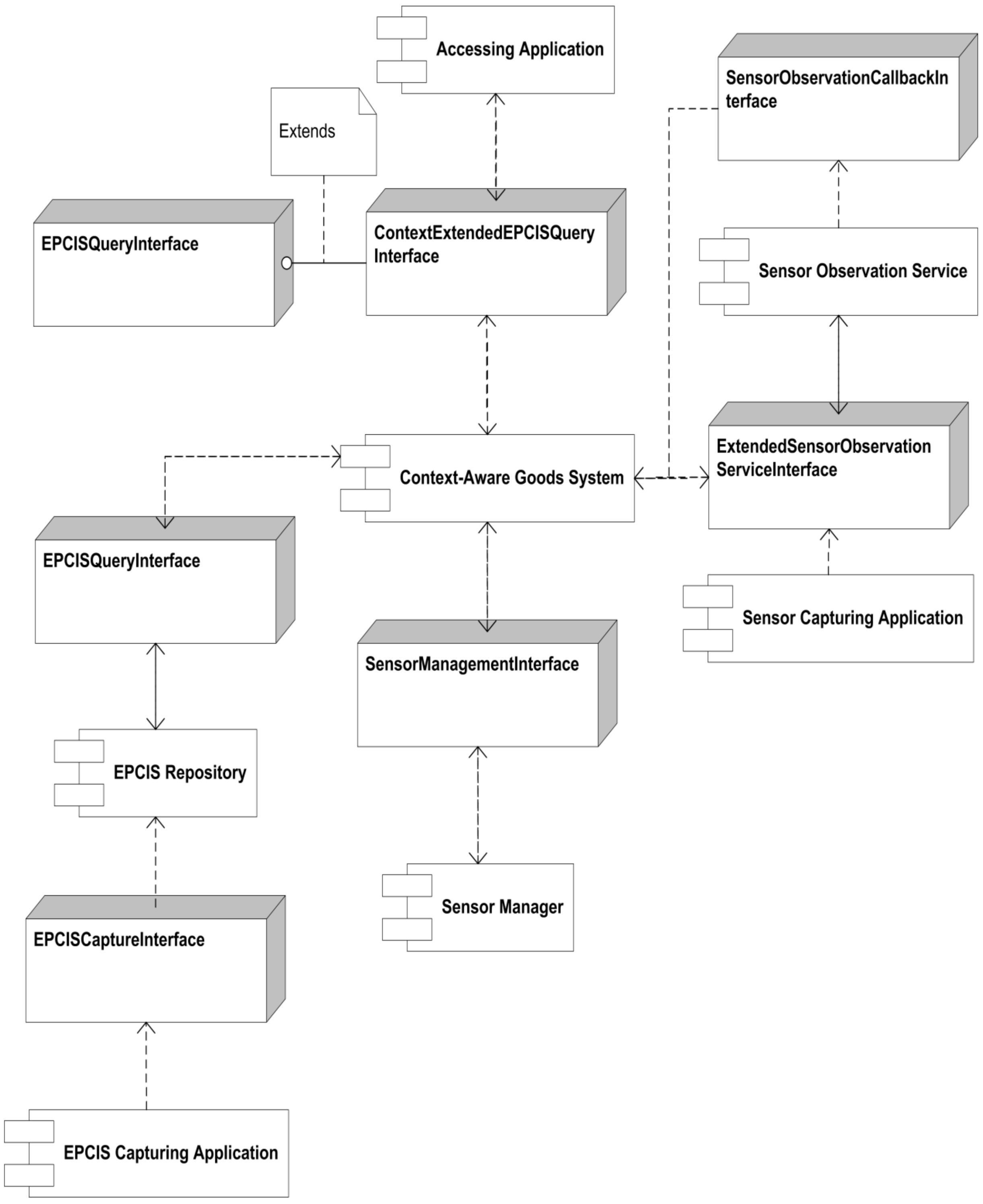

Accessing Application
Extends
SensorObservationCallbackInterface
EPCISQueryInterface
ContextExtendedEPCISQuery Interface
Sensor Observation Service
Context-Aware Goods System
ExtendedSensorObservation ServiceInterface
EPCISQueryInterface
Sensor Capturing Application
SensorManagementInterface
EPCIS Repository
Sensor Manager
EPCISCaptureInterface
EPCIS Capturing Application

Figure 27 Development view

15.5.3.1 ROLES

This section will describe each role in Figure 27 in detail. The roles are meant to be filled by software systems. One software system could fill several roles, or there could be one software system for each role. Figure 28 shows how a role is modeled in the development view.

Hardware/Software Role

Figure 28 The role component syntax

ACCESSING APPLICATION

This would be a software system, e.g. an Enterprise Resource Planning or a Customer Relationship Management system, dependent on information about goods. It is the same *Accessing Application* as EPC AF describes. The application will talk to the *CAGS* (Context-Aware Goods System), through the *ContextExtendedEPCISQueryInterface*.

CONTEXT-AWARE GOODS SYSTEM

This is the core of the software system. Here all sensors are mapped to EPCs with a time span. Queries for sensor observations for EPCs would come from an *Accessing Application* and be mapped to sensors before the *Sensor Observation Service* is queried. Queries for EPCIS Events will also go through this system to an *EPCIS Repository*.

EPCIS REPOSITORY

The repository storing EPCIS Events, explained in EPC AF. Used by the *EPCIS Query Interface* to retrieve information about what, where, when, and why events happened to the goods. More information can be found in Appendix B.

EPCIS CAPTURING APPLICATION

This role is described in EPC AF, and the job is to create EPCIS Events from actual readings of EPCs. This means adding business information to actual observations. More information can be found in Appendix B.

SENSOR OBSERVATION SERVICE

This role is from the SWE framework, and contains actual readings from sensors. It could e.g. contain temperature readings over time from a temperature sensor. It also contains information about a sensor explained in the notation SensorML. This information could be such as correctness, type of information measured and intervals for measurement. The sensor readings are specified in the O&M (Observations & Measurements) notation. More information can be found in Chapter 9.

SENSOR CAPTURING APPLICATION

This application is sending sensor readings to the *SOS* (*Sensor Observation Service*). This role is the software link to the hardware, transforming data coming from different readers to observations sent through the *ExtendedSensorObservationServiceInterface* to the *SOS*.

SENSOR MANAGER

This system maintains sensors; adding new sensors, deleting sensors and editing sensors. It also explains to *CAGS* what EPCs are connected to which sensors and the time span of the relation.

15.5.3.2 INTERFACES

This section will describe each interface in Figure 27 in detail. The interfaces are made to standardize the communication between the different roles. Even though one software system could fill several roles, excluding the need for some interfaces, it does not remove the need for interfaces because all the roles might be implemented separately. Figure 29 shows how an interface is modeled in the development view.

Interface (EPCglobal, SWE and IGS standards)

Figure 29 The interface component syntax

CONTEXT EXTENDED EPCIS QUERY INTERFACE

This interface defines how the *Accessing Application* interacts with *CAGS*. It extends the *EPCIS Query Interface* defined by EPCglobal with context specific functions. The reason for this extension is to enable the coexistence of context-extended systems and pure EPCIS systems. *Accessing Applications* not aware of the new extension would still be able to use it, just not the extended part.

EPCIS QUERY INTERFACE

This interface is from EPCIS as the name implies, and is used by *CAGS* to retrieve information from the *EPCIS Repository*. More information can be found in Chapter 7 and Appendix B.

EPCIS CAPTURE INTERFACE

This interface is also from EPCIS and is used by the *EPCIS Capturing Application*. More information can be found in Chapter 7 and Appendix B.

EXTENDED SENSOR OBSERVATION SERVICE INTERFACE

This interface describes how *CAGS* can retrieve readings and sensor descriptions, by querying or subscribing. It also defines how the *Sensor Capturing Application* store readings from sensors. More information can be found in section 15.5.1.3, where this interface is described in detail.

SENSOR MANAGEMENT INTERFACE

This Interface describes how the *Sensor Manager* interacts with *CAGS*; creating, editing or deleting sensors and managing sensor and EPC relations.

SENSOR OBSERVATION CALLBACK INTERFACE

This interface defines how the *SensorObservationService* can contact *CAGS* with information about subscribing observations. The subscriptions come from *Accessing Applications* through *CAGS*, mapped to sensors before subscribed to the *SensorObservationService.*

15.6 SCENARIOS

This section will describe how two scenarios from section 13 are proposed solved by the architecture. Each scenario is first explained by a summary; then the proposed solution is presented using figures and text.

15.6.1 PROVING THE QUALITY OF FRESH FOOD (SC 1.3)

This scenario will use an EPC for a fresh food product, e.g. fish, to prove the quality of the ware. The Accessing Application could be a control machine placed in a store, allowing customers to check the quality of the fish before they buy it. The fish is wrapped in paper, with an RFID inside the price tag.

PROPOSED SOLUTION BY THE CONTEXT-AWARE GOODS SYSTEM

The solution is presented in Figure 30, and every step is described in detail.

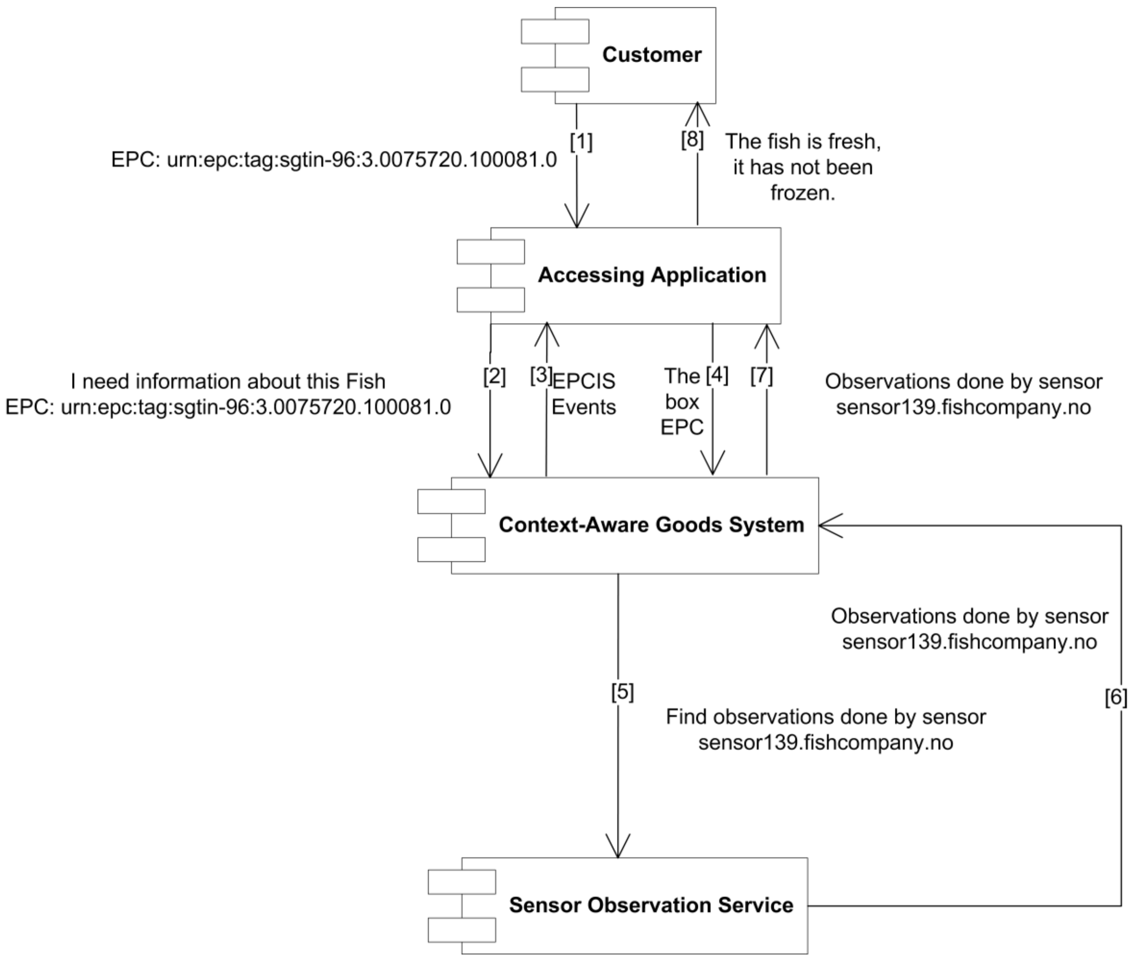

Customer
[1]
EPC: urn:epc:tag:sgtin-96:3.0075720.100081.0
[8]
The fish is fresh, it has not been frozen.
Accessing Application
[2]
I need information about this Fish EPC: urn:epc:tag:sgtin-96:3.0075720.100081.0
[3]
EPCIS Events
[4]
The box EPC
[7]
Observations done by sensor sensor139.fishcompany.no
Context-Aware Goods System
Observations done by sensor sensor139.fishcompany.no
[5]
Find observations done by sensor sensor139.fishcompany.no
[6]
Sensor Observation Service

Figure 30 Proposed solution of SC 1.3 by CAGS

The customer scans the fish giving the *Accessing Application* an EPC to investigate.

1. The EPC is sent to the *Accessing Application* calculating if the fish is fresh. To do this it needs information about how the fish has been treated before the customer scanned the EPC.
2. The *Accessing Application* uses ONS (Object Name Service). It sends the EPC to the *CAGS* and asks for EPCIS information. More information about ONS can be found in (EPCglobal 2007c).
3. Aggregation Events are retrieved explaining that the fish has been transported in a box, from time X to time Y.
4. This box's EPC is used to retrieve sensor information by sending the new EPC to *CAGS*. The *CAGS* finds relevant sensors based on the EPC and its relation to sensor URIs.
5. The sensor URI is sent to the *Sensor Observation Service* retrieving observations registered by this sensor.
6. The observations are sent to *CAGS*

7. The observations are sent to the *Accessing Application* and the information about temperature to time is used to control if the fish has ever been frozen. The *Accessing Application* verifies that the temperature has never been below 1 degree Celsius.
8. Information about the fish is sent to the customer, confirming the freshness of the fish.

15.6.2 AUTOMATIC ALARMS OF VIOLATIONS (SC 3.1)

This scenario will use EPCs to tag boxes of fresh fish transported from Trondheim to Arendal, two cities in Norway. The fish is transported by a trailer, containing a temperature-controlled box. Alerts of temperature violation can be sent to the driver's PDA, communicating with a central server using a regular mobile network.

PROPOSED SOLUTION BY THE CONTEXT-AWARE GOODS SYSTEM

The solution is presented in three figures. Every figure will present different aspects of the solution and they are all described in detail.

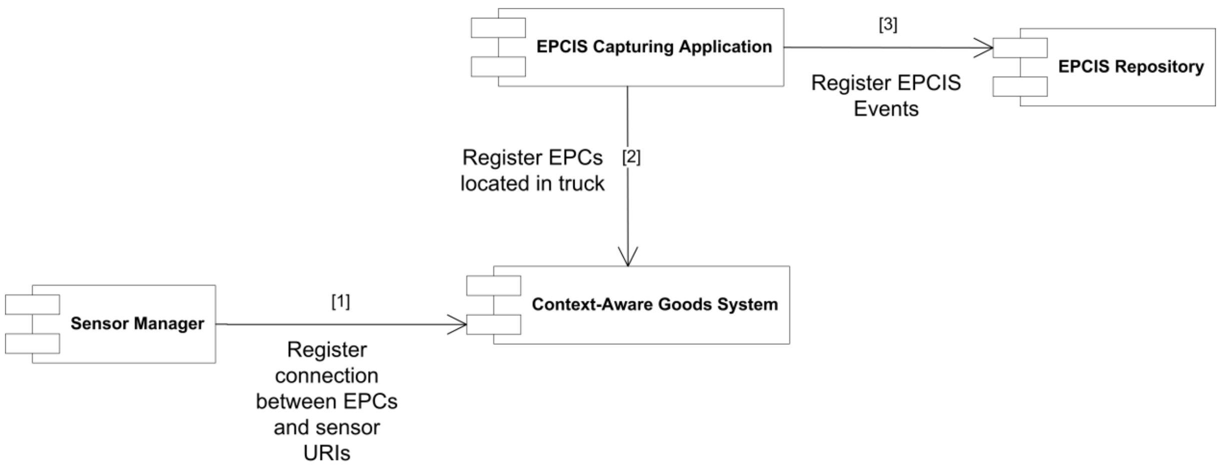

EPCIS Capturing Application
[3]
Register EPCIS Events
EPCIS Repository
Register EPCs located in truck
[2]
Sensor Manager
[1]
Register connection between EPCs and sensor URIs
Context-Aware Goods System

Figure 31 Proposed solution of SC 3.1 by CAGS; The management of sensors

Figure 31 describes the proposed solution when the sensors are set up and attached to the boxes, and later loaded on the trailer for transportation. In this figure the *Capturing Application* also plays the role of the *Sensor Manager*, both described in section 15.5.3.1.

1. When the boxes are made, the sensors are attached and the relation between the box's EPC and the sensor's URI has to be registered. This information is sent to the CAGS.
2. When the trailer loads the boxes, a gate at the exit registers this and the information is sent to CAGS because the trailer has a sensor as well. This needs to be updated so every EPC inside the trailer now have an extra sensor.
3. The capturing application also delivers EPCIS Events about the loading to an EPCIS Repository.

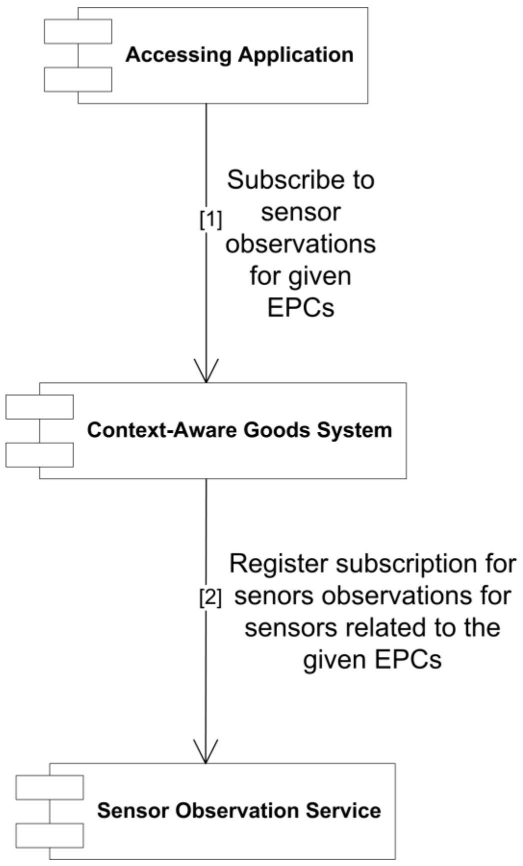

Accessing Application
[1]
Subscribe to sensor observations for given EPCs
Context-Aware Goods System
[2]
Register subscription for senors observations for sensors related to the given EPCs
Sensor Observation Service

Figure 32 Proposed solution of SC 3.1 by CAGS; The subscription

Figure 32 describes the proposed solution when the *Accessing Application* watching the goods subscribes for information about violations.

1. The *Accessing Application* controlling driver alerts subscribes to information about all the EPCs loaded onto the trailer.
2. The *CAGS* registers this subscription with the *Sensor Observation Service*

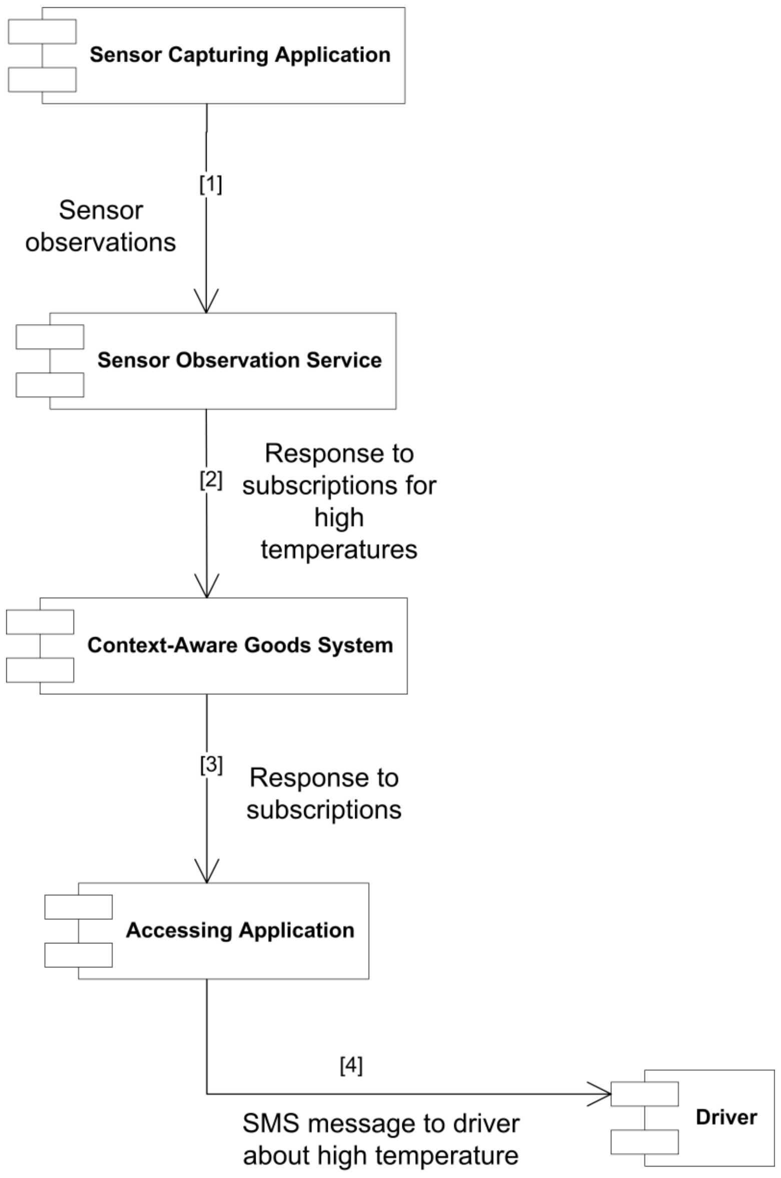

Sensor Capturing Application
[1]
Sensor observations
Sensor Observation Service
[2]
Response to subscriptions for high temperatures
Context-Aware Goods System
[3]
Response to subscriptions
Accessing Application
[4]
SMS message to driver about high temperature
Driver

Figure 33 Proposed solution of SC 3.1 by CAGS; The alert

Figure 33 describes the proposed solution when sensor readings are received by the system, eventually warning the driver about the temperature being too high.

1. A *Sensor Capturing Application* receives information from a sensor on a fish box and sends this to the *Sensor Observation Service* (SOS).
2. The *SOS* checks the sensor against subscriptions and finds a subscriber. The expression is checked and a violation is detected. The *SOS* sends this information to *CAGS*.
3. The *CAGS* maps the sensor to EPCs and finds the subscriber, sending the observation as an alert.
4. The *Accessing Application* receives the observation and a SMS message telling the driver to check the temperature-controlled box is sent. The temperature registered

by the sensor is also sent, to give the driver information to work with when fixing the problem.

15.7 SOFTWARE ARCHITECTURE RATIONALE

The important quality attributes for this system are: Security, Privacy, Scalability, Modifiability and Performance. This thesis has tried to make the software architecture to best suit all of them, but there are bound to be trade-offs between them. The next sections present how every attribute has been incorporated into the software architecture.

15.7.1 SECURITY AND PRIVACY

These two attributes have been given much focus because of their importance. The architecture itself is only a proposed solution, and the real focus comes when the implementation is done. To ensure privacy, security has been given attention as shown in Figure 25. Here the authentication and authorization of accessing applications are shown. This view only emphasizes the need for this to be solved during implementation. How, by whom, and what standards to use, the software architecture does not state.

15.7.2 SCALABILITY AND PERFORMANCE

By making the system scalable, this system needs to have high performance. This is because of the vast amount of sensors and id-tags on one side, and accessing applications on the other side. Figure 27, the development view, shows the different parts and roles of the system. All these roles can be duplicated, making it highly scalable. Retailers may have one of each, making all requests go through one central server park. Other retailers might have several stores in many countries, requiring several server parks in different regions. To prevent the need of several central servers, it is important to perform as much computation as possible at the edge of the system. The proposed software architecture is also based on elements of already developed systems, emphasizing scalability.

15.7.3 MODIFIABILITY

Modifiability is important because of the ever-changing aspect of the system. Sensors today will not be the same as sensors developed in the future. The context information considered interesting will also change, and it is important to be able to evolve with these changes. To be able to modify the system roles and interfaces have been a focus, shown in Figure 27. Standards emphasizing modifiability are also used, e.g. SensorML, EPCIS, and O&M.

Part IV Evaluation and Conclusion

16 EVALUATION

This chapter presents our evaluation of the software architecture presented in the previous part, and our answers to the research questions. It also presents a discussion of the findings, and an overall evaluation of the work done in this thesis.

16.1 ARCHITECTURAL EVALUATION

An evaluation of the architecture would normally be done, e.g., by an Architecture Tradeoff Analysis Method (ATAM) (Bass, Clements et al. 2005). It is difficult for us to come in contact with our stakeholders, given the time span of this thesis and the nature of the stakeholders. We can therefore not perform an ATAM at this time, but it would be useful to do so before an eventual implementation is started.

This section will evaluate the proposed software architecture with regard to the scenarios, functional requirements, and research questions. As both the scenarios and the functional requirements are based on the research questions, some of the information below will be similar, e.g., a scenario and a functional requirement. We have still decided to discuss each of them, even though some information might be repeated, to show how our software architecture can be used for each scenario, functional requirement, and research question.

The architecture is role and interface based, meaning that implementers could choose to have one application cover more roles at the same time. To support sharing of information with accessing applications, only the *Context Extended EPCIS Query Interface* described in Chapter 15 needs to be implemented. Implementing this interface would enable sharing of both EPCIS events and context information. The rest of the roles could theoretically be implemented by the same application. This would depend on the type of implementation, e.g. prototype or full implementation, and it would be necessary to consider the scalability of the final system when deciding how the roles should be divided.

The software architecture in this thesis is represented at a high level of abstraction. To be able to really implement this system, the whole SWE framework and EPCIS standard would need to be investigated. To give the implementer a full understanding of every detail at the bottom level, such as EPCIS events, sensor descriptions, and observations. The use of different roles to divide work could decrease the amount of information and understanding needed by each developer. Different teams could be assigned to work with the implementation of EPCIS, sensor management, and sensor observations.

16.1.1 SEPARATION OF CONCERNS

We decided to integrate two existing frameworks for the domains sensors and item tracking. This decision was made in order to take something good from each domain, and combine it into something new. There is no reason to reinvent the wheel if existing solutions can do the job. EPCIS is starting to get a wider use, and could easily become the de-facto standard for retailers. Because of this wide use the benefit of integration with EPCIS is obvious. It will be much easier for businesses to adapt the software architecture from this thesis if they are already using EPC tagged items and EPCIS.

The EPCIS integration was done by adding methods to the *EPCIS Query Interface*. By taking this approach the Object Name Service (ONS) and EPCIS Discovery Service from EPCglobal could still be used to find repositories with information about an EPC. If the Accessing Application retrieving information is pure EPCIS, it would still be able to use the new system, just without the new information available. Accessing Applications with context-awareness would be able to look up EPCs and investigate if there are only EPCIS Events or context information as well. In this way both types of Accessing Applications can access and retrieve information, and they will only get information they know how to handle.

An EPCIS repository is used to store EPCIS Events, and SensorML and O&M is used for context information. SensorML is used to describe a sensor, with information about accuracy, time interval, and type of values measured. O&M is used to describe actual readings or observations. Both O&M and SensorML are XML descriptions, defined in the SWE framework. This enables the two domains to evolve separately, and be the best at their part. If sensors change, SensorML would change (if needed), without the need of changes to EPCIS. The O&M could also change without any impacts on EPCIS and SensorML. However, both SensorML and O&M are made to be highly modifiable, reducing the need for changes to the two standards themselves.

We have also decided to have separate repositories. There will be one EPCIS repository for EPCIS events, another for storing SensorML models and SensorML instances, and one for observations. It is also possible to have separate repositories for SensorML models and actual sensor instances. The same is possible for O&M types and actual observations.

16.1.2 SCENARIO EVALUATION

This section evaluates the architecture with regard to the scenarios presented in Chapter 13. Each scenario type is presented, and an explanation on how the software architecture could enable the scenario is given.

QUALITY DETERIORATION (SC 1)

Scenario SC 1.1 on automatic and dynamic expiration dates is not solved directly by this software architecture, because the scenario is based on a tag placed directly on the goods. This tag is thought of as being a communicator to the end user, and therefore does not directly use Context-Aware Goods System (CAGS). The tag is preconfigured before the goods are put out into the shelves, with information about the current quality. After this the tag lives without communication with CAGS, or else the communication needed would be a problem, both in a security and privacy perspective.

Scenario SC 1.2 could be solved by a solution implementing the proposed software architecture. Context information about the food would be stored in an observation repository, and subscriptions can be used to retrieve information about undesirable values. With this solution in place, users retrieving this information on food quality would need to build a software system reacting to the information and correctly alert persons or other software systems, e.g. by sending a text message to the driver's cell phone.

Scenario SC 1.3 depends on a temperature history of the food. This could be obtained using our architecture, as context information would be sensed and registered together with the food's EPC. Accessing Applications could then lookup EPCs and check for temperatures outside of the accepted range.

TEMPERATURE PROFILES (SC 2)

Scenarios SC 2.1 and SC 2.2 depend on the same information. They require a temperature profile showing how the temperature varies in a given space. To create such a profile it is necessary to know how the sensors are positioned relative to each other. This can be achieved in two different ways with our proposed software architecture. If all the temperature sensors are mounted rigidly in the space, a sensor system can be modeled in SensorML, and an offering which includes the observations from the individual sensors as well as relative positioning among them could be provided. This information could then be retrieved either directly from the SOS, or through the Context Extended EPCIS Query Interface. To use the last method it would be necessary to either identify the space with an EPC or to have the EPC of an item that has been in that space. This is because our solution retrieves sensor information related to given EPCs.

The other way to achieve these scenarios would be to include temperature sensors on the goods stored in the space. This could either be on each item or, e.g., on each pallet. By using our Association Event, which extends the EPCIS event, relative positioning among these items is possible. An accessing application could then query the *Context Extended EPCIS Query Interface* for information on relative positioning among the items, and for the observations from the sensors mounted on the items. Using this information, an accessing application could detect cold spots, map the need for insulation of goods, and calculate how many sensors are needed in a given space using historical data.

Scenario SC 2.3 depends on two sensors mounted directly on the items. This could be achieved using SensorML to specify a sensor with two detectors, one inside and one outside of the insulation. An accessing application could then query the *Context Extended EPCIS Query Interface* for observations from this sensor, and observe differences in temperature from the two detectors. This could be used to determine the amount of insulation required.

SENSOR COLLABORATION AND INTELLIGENT GOODS (SC 3)

These scenarios describe sensors which increase in sophistication. In the first scenario the sensors give alerts when certain conditions are not met (SC 3.1). The second scenario has sensors which automatically takes action if these conditions are not met, without consideration for other sensors in the same space (SC 3.2). In the third scenario the sensors collaborate on these actions (SC 3.3). The actions taken would, e.g., be to automatically control the climate control in the space.

One way to solve the last two scenarios, SC 3.2 and SC 3.3, would be to have intelligent sensors communicating between each other, and with the climate control system. This could be done without using our proposed Context-Aware Goods System, but would require much logic implemented on the sensors themselves, especially for the third scenario.

The scenarios could also be solved by CAGS. This would require a central application receiving the information from the sensors and carrying out the actions, the sensors would not carry out the actions themselves. The first scenario describes sensors alerting when something is wrong. This can be achieved with our subscription mechanism, where accessing applications can subscribe to observations from the sensors. The other two scenarios would then depend on the accessing application taking actions on the information received. It could either react on observations received from subscriptions, or it could actively collect the information needed through the *Context Extended EPCIS Query Interface* and then decide what it should do.

We believe a good solution could be an application which periodically asks for observations to adjust the environmental controlling systems, based on information from all sensors in the space. The application would also subscribe to information on observations that are out of range, enabling alerts when something is wrong. This solution would not actually make the goods collaborate, but the actions taken would depend on information from sensors placed on all the goods. Actual sensor collaboration would, as mentioned above, require very intelligent sensors, and are not in the scope of our software architecture.

HIERARCHY OF GOODS WITH SENSORS (SC 4)

Automatic aggregation of goods can be done by setting up predefined gates or readers at given spots. For example at a packaging station, a reader could be set up to read every package departing. Every package could be read for the package's EPC and the EPCs inside the package. Then the reader sends all this information to the capturing application, automatically storing this as an aggregation event. Aggregation events are built-in to EPCIS, and as our proposed architecture depends on EPCIS this could be achieved.

As aggregations in EPCIS can be identified by parent EPCs, e.g. an EPC on a pallet, it would be possible to connect sensors to the aggregated item using our solution. The *Context Extended*

EPCIS Query Interface provides methods to query for sensor information related to EPCs. This enables queries for sensor observations of an aggregation instead of querying for each item in the aggregation. Because EPCIS Aggregation Events supports multiple levels in the hierarchy by aggregating parent EPCs into new aggregations, each identified by an EPC, sensors could be connected to any level in the hierarchy. This gives accessing applications the opportunity to retrieve sensor observations at the desired level in the hierarchy.

Aggregating items could greatly reduce the amount of queries needed. If e.g. a pallet has a temperature sensor, EPCs placed on that pallet could be connected to that sensor. If these items were aggregated together, it would only be necessary to query for the temperature of the parent EPC, and not for every item on the pallet. If it was a temperature sensor in the room instead of on each pallet, the benefit would be the same. Both the EPCs of the items and of the pallets could be connected with the sensor in the room, and it would be possible to query both for item EPCs and pallet EPCs.

The connection between EPCs and sensors described above would be done in CAGS as described in Chapter 15.

PROXIMITY CONTROL (SC 5)

Scenario SC 5.1 and scenario SC 5.2 requires similar technology. They both require the knowledge of how goods are placed relative to each other, how close they are, and what kind of goods is stored. This could be realized by the proposed association event, which provides information on relative positioning among items. This event would have to be created either manually when goods are stored, or by having sensors on every ware and a capturing application receiving information from the sensors and generating the event. The subscription methods could be used to receive information if the wrong kinds of goods are stored together. An alert could be raised when receiving information from the subscriptions.

We have previously mentioned that sensor systems can be modeled with information on the relative positions between sensors. This would not be useful for this scenario, as the scenario focuses on storage facilities where the goods stored change with time. Such sensor systems are better suited for sensors mounted rigidly.

16.1.3 REQUIREMENTS EVALUATION

This section presents the functional requirements from Chapter 14, and shows how the proposed software architecture can be used to solve them.

FR 1. The system must support data from different sensor types
   a. It must support configuration of new sensor types from a generic sensor type

This requirement is proposed solved using SensorML and O&M from the SWE framework. Our software architecture uses SensorML to describe the sensors used and O&M to describe the observations. Both these standards are generic, and can be used to describe just about any sensor type and observation type. New sensor types not considered at implementation will not be a problem, as SensorML is a generic XML-based language, and it does not require the sensor type to be specified before implementation. New sensor descriptions in SensorML would have to be written for each new sensor type used, but this does not impact the rest of the system. Using repositories to store the SensorML models, it would be easy to include new sensor models. If SensorML gets a wide use, it could be possible that sensor manufacturers would provide this information.

FR 2. The system must be able to receive real-time data from readers
  a. Sensor data combined with a time aspect
  b. Identification of items

This requirement is proposed solved by integrating the EPCIS and SWE frameworks. In SensorML sensors are identified by URIs, and in EPCIS items are identified by EPCs. Sensor data combined with a time aspect could be retrieved from the SOS, and identification of items from EPCIS. The proposed CAGS combines relates items to sensors, and provides a common interface to retrieve item and context information.

FR 3. The system must extend a logistic information system based on identification with a context aware information system.

This requirement is proposed solved by integrating EPCIS with SWE in our CAGS. EPCIS is used to track information on items identified by EPCs, and SWE is used to describe and share context information. The proposed *Context Extended EPCIS Query Interface* extends the standard *EPCIS Query Interface* with additional methods for context information.

FR 4. The system must be integrated with EPCIS
  a. Generate EPCIS Events
  b. Submit EPCIS Events through an EPCIS Capture Interface

This requirement is proposed solved by extending the *EPCIS Query Interface* to support context information. CAGS appends a context aware system to a fully functional EPCIS system. In this way EPCIS is kept as normal, with the capturing application and other roles intact. EPCIS Events will be generated by a capturing application, and they will be sent through a capture interface in the normal way.

FR 5. The system must support a hierarchical structure of items
  a. A generic model for hierarchy

The proposed software architecture supports standard EPCIS events. One of these events is the Aggregation Event, which can be used for hierarchical structures. This requires the use of EPCs for identification of items, which is used in our architecture.

FR 6. Items must have an unique identification (URI)

This is solved by requiring all items to be uniquely identified by an EPC, and an EPC is an URI. This comes naturally when supporting EPCIS. Sensors are also uniquely identified in SensorML by URIs.

FR 7. The system must support zero to many sensors on items

a. Identification of each sensor.

This is proposed solved in CAGS by mapping EPCs to sensor URIs. Every sensor is identified with an URI and described with SensorML. Our system maps EPCs to sensor URIs in a many-to-many relationship.

FR 8. The system must support relative positions of items

a. Relative positions to other items in a given space
b. A generic model for relative position of items

This is proposed solved by extending the EPCIS Event base type with an Association Event. This event describes the relative distance between two items. The event contains the EPCs of the two items and a distance vector describing the direction and distance from one item to the other. This event could be created manually by observing the items, or automatically from observations retrieved from sensors on the items able to sense distance to other sensors or items.

FR 9. The system must have a generic alert mechanism

a. Enable alarms and alarm handling
b. Able to define sensor types to watch, and laws for these types
c. Able to execute a procedure if laws are broken.

The proposed solution for this is the *subscribe* method available from the *Context Extended EPCIS Query Interface*. This requires users interested in receiving alerts to subscribe to each sensor he wants to monitor. With EPCIS it is possible to have many different EPCIS repositories at different locations, and this could also be done for CAGS. In such a situation it would be necessary to register subscriptions at all locations that could possibly have this information. This could seem a little bothersome, and it would be easier with one centralized location, but it is done to support scalability. This would have to be considered at each implementation.

FR 10. The system must support sharing of sensor information using SensorML

Support for sensors and observations are included in other requirements as well, e.g. FR1 and FR 3 explained above. FR 10 covers how information about the sensors themselves should be shared. This requirement is a little vague, as it does not specify what sensor information is. It could be information about the sensor itself, i.e. description of a sensor, or it could be information coming from the sensor, i.e. observations. The requirement also specifies that SensorML should be used for sharing, but SensorML is a modeling language used to describe sensors, not to share information. This ambiguity is due to a limited knowledge of the SWE standards when the requirements were written.

The software architecture we have proposed uses the SWE framework for context information. This means that SensorML is used to describe sensor information, O&M is used to describe sensor observations, and Sensor Observation Service is used to share this information. Our proposed *Context Extended EPCIS Query Interface* has methods to retrieve this information, *describeSensor* for sensor information and *getObservation* for observations.

16.2 RESEARCH QUESTIONS EVALUATION

This section presents the research questions presented in Chapter 2, and gives our proposed answers. Because many of the functional requirements above are based on these questions, some of them have already been answered. We will still answer each of the research questions here.

RQ 1. How can an electronic ID tracking information system be extended with sensor information?

There are many different ways to extend a tracking system with sensor information. We propose to solve this by integrating two frameworks, one for item tracking and one for sensor information. Chapter 15 presents a software architecture showing how this can be done. We have used the EPC Architecture Framework, and especially EPC Information Services, from EPCglobal as the tracking system. Our proposed software architecture extends the *EPCIS Query Interface* with methods to retrieve sensor information. By this approach systems based on our architecture can coexist with standard EPCIS systems. Accessing applications aware of our extension can utilize the new methods through the *Context Extended EPCIS Query Interface*, while other applications can use this interface as a standard *EPCIS Query Interface* to retrieve EPCIS Events.

We propose to use the Sensor Web Enablement (SWE) framework for the sensor information and observations. This includes SensorML for sensor descriptions, Observations & Measurements for observations, and Sensor Observation Service to share this information. Our software architecture is built so this could be exchanged for something else, but we believe that SWE is a good choice as it is an open global standard and it provides a generic way to describe both sensors and observations.

RQ 2. How to achieve relative positioning of one tagged unit with regards to other tagged units (next to, under, above)?

This can be done by using sensors able to track proximity of other sensors, and storing this information using EPCIS Events. We have proposed an extension to the EPCIS Event called Association Event which describes how two uniquely identified items (using EPC) is placed relative to each other. The association event describes this relative position using a vector, which includes direction and distance.

Relative positioning can also be achieved without proximity sensors. The EPCIS Association Event could be generated by an application based on manual input from a human user instead of sensor observations. This event would then be treated the same as automatically generated events.

RQ 3. How to build a generic software architecture towards sensors and tags that can be used in many different situations?

We have tried to solve this by designing our software architecture based on known standards which emphasize generalization. The use of the EPC AF and SWE frameworks, which are both built to be generic, will help our solution to be generic. We believe that our solution might work for many different situations, because of the focus on being generic, but it is difficult to predict future use.

This research question is partly answered, as it is generic towards sensors but only partly generic towards RFID tags. We require that the tags are identified by an EPC, which limits the tags used. Our architecture does not require that the tags used follow an EPCglobal tag-air interface, e.g. the UHF Gen 2 Class 1 standard used today (EPCglobal 2007b), and in theory any tag type can be used as long as it carries an EPC. Sensors are described by SensorML, which provides full support for generic sensors. Because of this we consider the software architecture for the most part to be generic.

RQ 4. How to ensure scalability in an ID tracking information system extended with sensor information?

Our proposed software architecture is as mentioned above built on the EPC Architecture Framework, which is built to scale. Our architecture integrates the SWE framework with EPC AF, and inherits many of the same qualities as EPC AF. Every role in the architecture can be duplicated, and it is up to the implementer to decide how many instances to create. We believe that the possibly large number of sensors and tags requires a decentralized information system. In our architecture this is solved in the same way as in EPC AF by giving the opportunity to have many different repositories, instead of just one central. This means that accessing applications will have to access more than one service in order to retrieve all relevant information, but this is necessary in order to ensure good scalability.

We believe that the answer to this research question is to spread the information on different repositories, and to make decisions at the edge of the network where possible. If every possible sensor observation and RFID tag detection should go through the same central server it would have a big impact on scalability. Both the EPC AF and SWE have the same focus of taking decisions at the edge, e.g. by filtering false reads and only storing relevant information.

RQ 5. How can a system with sensor information be integrated with EPC and EPCIS from EPCglobal?

We propose to solve this by extending the *EPCIS Query Interface* with methods providing access to sensor information. This is possible because of the focus on extensibility in the EPCIS standard, providing the opportunity to extend the interfaces in the standard. This is explained as an answer to research question 1 above.

RQ 6. How can SensorML be integrated with EPCIS?

This research question is partly answered, as SensorML is used together with EPCIS in our software architecture, but it is not enough to use SensorML alone. SensorML is, as mentioned above, used in the SWE framework to describe sensor information. We provide SensorML descriptions through the *describeSensor* method in our *Context Extended EPCIS Query Interface*, but a Sensor Observation Service (SOS) from SWE is needed to actually store and retrieve this information. The CAGS, which implement the *Context Extended EPCIS Query Interface*, uses SOS to retrieve sensor descriptions whenever needed. The sensor descriptions are then sent to an Accessing Application as XML, following the SensorML specification.

16.3 DISCUSSION

We believe that our proposed software architecture can be used to solve the issue of extending an electronic ID tracking system with sensor information. We decided to integrate two frameworks leading in their areas, EPC AF for item tracking and SWE for sensor information. This was done because we believe that it is much easier to start using a system if it is based on standards already known. If a company is currently using EPC-tagged items and EPCIS to track their movements, it would be much easier for them to extend this system with sensor information instead of adopting an entirely new system. The same point can be made for sensor information. If we had developed a new sensor description language, it would be difficult to get sensor vendors to provide descriptions of their sensors in our language. They might have done this if our language had gained wide use, but in the meantime companies interested in implementing our software architecture would have to write their own sensor descriptions for each sensor type.

The main contribution of this thesis is the software architecture integrating EPC AF and SWE. It took some time to figure out how we could best integrate item tracking and sensor information, but once we decided to use EPCIS and SWE, the solution was not that difficult. We have tried to design a software architecture using as much from the two frameworks as we could; only adding what is necessary to combine them. This was done to be as compatible as possible with existing software systems already using either of the two frameworks. We also wanted the advantages of using open global standards, e.g. full access to the specifications, an open development process for future versions, and global interoperability. Even if the resulting software architecture is simple, it took time to think of the solution because both specifications had to be evaluated and investigated. The result of the process is a software architecture extending EPCIS, ready to be implemented as an extension to EPCIS without impacting existing implementations.

One problem encountered during the work on this thesis, was the large amount of information and knowledge necessary to be acquired. Both frameworks used are large and complex, and it took quite a while to achieve enough understanding of these to proceed with the work. Even though we knew about EPCIS from previous work, we had to figure out how it could best be extended. SWE was completely new to us, as well as sensor technology in general. As mentioned in Chapter 11, the documentation of SWE was relatively difficult to read, and this was one reason that it took quite some time to investigate the framework. The decision to use SWE was also taken relatively late in the process, resulting in a short period of time to develop the software architecture. We did not have as much time to evaluate and improve the description of the software architecture as wanted, but we still believe that the architecture provided is a good solution to the problem at hand.

The approach taken with the software architecture focuses on using EPCIS as much as possible, and adding context information from SWE. If we had more time, or for a later project, it could be interesting to try to develop a solution the other way around, with SWE at the center and extending with EPCIS. RIFD readers could be considered as sensors and modeled by SensorML, and item observations could be described using standard O&M used for sensor observations. Then either the Sensor Observation Service or the Sensor Planning Service from SWE could be extended with EPCIS methods providing EPCIS events. This approach would lose the benefit our approach has by utilizing existing EPCIS implementations, but it would have been interesting to further explore these possibilities. Perhaps it could be possible to use existing standard EPCIS implementations for companies not interested in sensor readings, and only implement the SWE standards for those interested in additional context information. If these companies already had implemented an item tracking solution using EPCIS they could probably reuse some of this, but they would need to change at least parts of their systems.

Overall we believe that the choice of using EPCIS as the basis and extending with SWE was a good solution. It seems that the EPC standards are more used today, while SWE is relatively new and have not gotten that much attention. It also seems that the SWE standards are not quite finished. There are many notifications of future changes and potential problems with various parts of the standards in the standard documents. It is of course still possible to use them at this point, but we did not consider it wise to use them as the basis for our software architecture. The reason for using EPCIS is also related to the nature of the problem description of this thesis. The focus is on tracking items, in this case fresh food, and collecting context information as they are tracked. We believe that it is better to implement a tracking system and later extend with context information, than to implement a sensor network and later extend with the possibility to track items.

17 CONCLUSIONS AND FUTURE WORK

This chapter concludes our thesis. The first section presents concluding remarks, before the second briefly discusses future work.

17.1 CONCLUSIONS

Tracking of goods and related context information are becoming more and more important. We believe that in a few years this will be regulated by the government. All fresh food sold can be traced back to its place of origin, and information on how it has been treated on the way is easily available. There are also many other benefits for businesses able to track and trace their products through their supply chains, thus it will be motivating to implement such solutions even where it is not mandatory.

We have found out that a lot of work has been done on item tracking, and especially after EPCglobal was established and global standards became available. There are also many projects on sensor information related to fresh food, but we do not see the same focus on standardization there. The SWE standards from OpenGIS might take the same position for sensors and sensor information as EPC has done for electronic identification.

We do not think that it is a good idea to combine these standards into one, as they cover very different areas. But we do believe that solutions combining the information they provide are very interesting. By having two separate standards it is also possible for them to develop independently as the technology advances and new and different needs arise.

This thesis has shown some of the possibilities arising when a context-aware tracking solution is available through a set of scenarios. These are of course only examples, and the possibilities are many. It has also presented an evaluation of one standard framework for each of the two areas item tracking and context information, together with a software architecture where these frameworks are integrated. The evaluation in Chapter 16 shows that the research questions presented in Chapter 2 can be answered using our proposed software architecture.

17.2 FUTURE WORK

This software architecture is an example of how a tracking system can be extended with context information. Before implementing this architecture we believe that it is important to consult with the different stakeholders. We have stated who we consider as stakeholders for the solution, and have tried to anticipate their needs, but due to time constraints we have not been in contact with them. The information we have gained have been through literature studies and discussions with supervisors, and a more thorough stakeholder analysis is necessary.

It is also necessary to implement a proof-of-concept solution. We have shown a software architecture that can be used for this, but in order to get companies interested it is important to have a working prototype. This architecture is at a high level of abstraction, and a detailed design phase together with a pilot implementation would take this project one step further.

The work being done by the two standards organizations should be followed. As both of the frameworks we have used are in version 1.x, it is important to remember that they can be changed in the future. As more and more companies start using these frameworks, the organizations will get feedback on how they are performing together with suggestions for improvements. There could be issues with the current specifications that neither we nor the standards organizations are aware of at this time. We do believe that this is also one of the benefits of using standard specifications as the basis for our solution. More users means more feedback, and we believe that both frameworks will evolve and improve in the future.

PART V Bibliography and Appendices

BIBLIOGRAPHY

Albretsen, S. and M. A. Larsen (2007). EPC Information Services: Usage and Value Scenarios of EPCIS. Department of Computer and Information Science. Trondheim, Norwegian University of Science and Technology.

Ari, J., L. R. Ronald, et al. (2003). The blocker tag: selective blocking of RFID tags for consumer privacy. Proceedings of the 10th ACM conference on Computer and communications security. Washington D.C., USA, ACM.

Asher, C., G. Morgan, et al. (2007). "EPCIS Frequently Asked Questions." Retrieved April 01, 2008, from http://www.epcglobalinc.org/standards/epcis/epcis_1_0-faq-20070427.pdf.

Bass, L., P. Clements, et al. (2005). Software Architecture in Practice, Second Edition, Addison-Wesley.

Botts, M., G. Percivall, et al. (2007, December 28). "OGC® Sensor Web Enablement: Overview And High Level Architecture." OGC White Paper 3.0. Retrieved Mars 01, 2008, from http://portal.opengeospatial.org/files/?artifact_id=25562.

Brock, D. L. (2003, June 1). "Fresh Food – Dynamic Expiration Dates Using Auto-ID Technologyand Analytic Shelf Life Models." White Paper Retrieved April 24, 2008, from http://www.autoidlabs.org/uploads/media/MIT-AUTOID-WH019.pdf.

CAEN. (2008). "CAEN Homepage." Retrieved April 10, 2008, from http://www.caen.it/.

Chen, G. and D. Kotz (2000). A Survey of Context-Aware Mobile Computing Research. Technical Report, Darthmouth College.

Culler, D., D. Estrin, et al. (2004). "Guest Editors' Introduction: Overview of Sensor Networks." Computer **37**(8): 41-49.

Dey, A. K. and G. D. Abowd (1999). Towards a better understanding of context and context-awareness. Technical Report, Georgia Institute of Technology.

Ding, Z.-h., J.-t. Li, et al. (2007). A Filter Design of RFID Middleware in the Progress of Updating Barcode to RFID. RFID Eurasia, 2007 1st Annual.

Engels, D. W. and S. E. Sarma (2002). The reader collision problem. IEEE International Conference on Systems, Man and Cybernetics 2002.

EPCglobal (2005). Object Naming Service (ONS) Version 1.0. EPCglobal Ratified Specification, EPCglobal Inc.

EPCglobal (2007a). EPC Information Services (EPCIS) Version 1.0.1 Specification. EPCglobal Ratified Specification, EPCglobal Inc.

EPCglobal (2007b). EPC Radio-Frequency Identity Protocols Class-1 Generation-2 UHF RFID Protocol for Communications at 860 MHZ - 960 MHz Version 1.1.0. EPCglobal Ratified Specification, EPCglobal Inc.

EPCglobal (2007c). The EPCglobal Architecture Framework Version 1.2. EPCglobal Ratified Specification, EPCglobal Inc.

EPCglobal (2007d). EPCglobal Tag Data Standards Version 1.3.1. EPCglobal Ratified Specification, EPCglobal Inc.

EPCglobal (2008a). The Application Level Events (ALE) Specification Version 1.1. EPCglobal Ratified Specification, EPCglobal Inc.

EPCglobal. (2008b). "EPCglobal Homepage." Retrieved Mars 24, 2008, from http://www.epcglobalinc.org.

Fowler, M. (2005). UML Distilled, Third Edition, Addison-wesley.

Gamma, E. (1994). Design patterns: elements of reusable object-oriented software. Reading, Massachusetts, Addison-Wesley.

Hanebeck, C. (2006). "Managing Data from RFID & Sensor-based Networks." White Paper Retrieved June 01, 2008, from http://www.globeranger.com/pdfs/futureoftheedge/GlobeRangerRFIDData.pdf.

Harmon, C. K. (2006). "The necessity for a uniform organisation of User Memory in RFID." International Journal of Radio Frequency Identification Technology and Applications **1**(1): 41-51.

Hawryszkiewycz, I. (2001). Introduction to systems analysis & design, Pearson Education.

Juels, A. (2006). "RFID security and privacy: a research survey." IEEE Journal on Selected Areas in Communications **24**(2): 381-394.

King, P. (2006). "Six Sigma and the Single Tag." RFID Journal Retrieved February 29, 2008, from http://www.rfidjournal.com/article/articleview/2102/1/82/.

Kruchten, P. B. (1995). "The 4+1 View Model of architecture." Software, IEEE **12**(6): 42-50.

KSW. (2008). "KSW Homepage." Retrieved April 10, 2008, from http://www.ksw-microtec.de.

Landt, J. (2005). "The History of RFID." IEEE Potensials **24**(4): 8-11.

Leaver, S. (2004, August 13). "Evaluating RFID Middleware." Tech Choices Retrieved April 05, 2008, from http://www.bauer.uh.edu/rfid/ForresterRFIDwave.pdf.

Liard, M. J. (2004, June). "The Co-Existence of EAS and RFID Technologies in Retail Environments: An Independent Perspective." White Paper Retrieved May 01, 2008, from http://www.adt.com/wps/wcm/resources/file/eb141f026a257fb/VDC_RFID_EAS-WhitePaper.pdf.

Logistikk & Ledelse. (2007). "Gilde, Intermec og SINTEF sammen om stort utviklingsprosjekt: RFID skal klare biffen " Logistikk og Ledelse Retrieved Mars 04, 2008, from http://www.logistikk-ledelse.no/2007/it/it01-03.htm.

Merriam-Webster. (2008). "Merriam-Webster Online Dictionary."   Retrieved Mai 2, 2008, from http://www.merriam-webster.com/dictionary/context.

Michael, K. and L. McCathie (2005). The pros and cons of RFID in supply chain management. International Conference on Mobile Business, 2005. ICMB 2005.

Montalbano. (2008). "Montalbano Homepage."   Retrieved April 01, 2008, from http://www.montalbanotechnology.com/.

Myhren, K. A. (2007). Personal communication with Kjell Arne Myhren from GS1 Norway. M. A. Larsen. Oslo.

Nahas, H. A. and J. S. Deogun (2007). Radio Frequency Identification Applications in Smart Hospitals. Twentieth IEEE International Symposium on Computer-Based Medical Systems, 2007. CBMS '07. .

Nofilis. (2007). "CrossTalk White Paper 2.0." White Paper Retrieved January 25, 2008, from http://www.nofilis.com/documents/CrossTalk_WhitePaper_2.0_EN.pdf.

O'Connor, M. C. (2005, January 18). "Gen 2 Finds a Path to ISO Approval." RFID Journal Retrieved Mars 10, 2008, from http://www.rfidjournal.com/article/articleview/1346/1/1/.

O'Connor, M. C. (2006a, July 11). "Gen 2 EPC Protocol Approved as ISO 18000-6C." RFID Journal Retrieved Mars 10, 2008, from http://www.rfidjournal.com/article/articleview/2481/1/1.

O'Connor, M. C. (2006b, January 31). "Tag Proposal Addresses Industry Needs." RFID Journal Retrieved Mars 10, 2008, from http://www.rfidjournal.com/article/articleview/2113/1/1/.

Open Geospatial Consortium (2007a). Observations and Measurements – Part 1 - Observation schema. S. Cox, Open Geospatial Consortium.

Open Geospatial Consortium (2007b). Observations and Measurements – Part 2 - Sampling Features. S. Cox, Open Geospatial Consortium Inc.

Open Geospatial Consortium (2007c). OpenGIS Sensor Model Language (SensorML) Implementation Specification. M. Botts, University of Alabama in Huntsville.

Open Geospatial Consortium (2007d). OpenGIS Sensor Planning Service Implementation Specification. I. Simonis, OpenGIS.

Open Geospatial Consortium (2007e). Sensor Observation Service. A. Na and M. Priest, OpenGIS.

Open Geospatial Consortium. (2008). "OGC Website." Retrieved May 23, 2008, from http://www.opengeospatial.org/.

OpenGIS. (2008, May 16, 2008). "SWE Common Data Model 1.1 Standards Working Group." Retrieved May 16, 2008, from http://www.opengeospatial.org/projects/groups/swecommonswg.

Posten. (2008). "Posten homepage." Retrieved Mars 01, 2008, from http://www.posten.no.

Potdar, V., P. Hayati, et al. (2007). Improving RFID Read Rate Reliability by a Systematic Error Detection Approach. RFID Eurasia, 2007 1st Annual.

Reichardt, M. (2003). "The Sensor Web's Point of Beginning." Retrieved May 15, 2008, from http://www.geospatial-solutions.com/geospatialsolutions/article/articleDetail.jsp?id=52681&pageID=1&sk=&date=.

Roberti, M. (2007, January 22). "Moving Forward on an HF EPC Standard." RFID Journal Retrieved Mars 12, 2008, from http://www.rfidjournal.com/article/articleview/2981/1/2/.

Rossen, E. (2008, February 22). "EU ser strengt på RFID og personvern." digi.no Retrieved Mars 17, 2008, from http://www.digi.no/php/art.php?id=511447.

Sarma, S. (2004). "Integrating RFID." Queue **2**(7): 50-57.

Satyanarayanan, M. (2003). "From the Editor in Chief [Of Smart Dust and Brilliant Rocks]." Pervasive Computing, IEEE **2**(4): 2-4.

SmartCode Corp. (2006). "SMARTCODE™ CORP. announces the world's first 5 cent RFID tag." Retrieved April 08, 2008, from http://www.smartcodecorp.com/newsroom/01-05-06.asp.

TimeTemp. (2008). "TimeTemp homepage." Retrieved Mars 14, 2008, from http://www.timetemp.no/.

Ukkonen, L., D. Engels, et al. (2005). Reliability of passive RFID of multiple objects using folded microstrip patch-type tag antenna. Antennas and Propagation Society International Symposium, 2005 IEEE.

University of Cambridge, AT4 wireless, et al. (2007, August 15). "High level design for Discovery Services." Retrieved December 04, 2007, from http://www.bridge-project.eu/data/File/BRIDGE%20WP02%20High%20level%20design%20Discovery%20Services.pdf.

Van Roy, P. and S. Haridi (2004). Concepts, Techniques, and Models of Computer Programming, MIT Press.

Vogt, H. (2002). Multiple object identification with passive RFID tags. IEEE International Conference on Systems, Man and Cybernetics, 2002.

Want, R. (2006). "An introduction to RFID technology." Pervasive Computing, IEEE **5**(1): 25-33.

Weinstein, R. (2005). "RFID: a technical overview and its application to the enterprise." IT Professional **7**(3): 27-33.

Xiao, Y., S. Yu, et al. (2006). "Radio frequency identification: technologies, applications, and research issues." Wireless Communications and Mobile Computing **7**(4): 457-472.

Yingping, C., W. Dong, et al. (2007). PTSP: a lightweight EPCDS platform to deploy traceable services between supply-chain applications. RFID Eurasia, 2007 1st Annual.

A ACRONYMS

CAGS	Context-Aware Goods System
EPC	Electronic Product Code
EPC AF	Electronic Product Code Architecture Framework
EPCIS	Electronic Product Code Information Services
HF	High Frequency
NFC	Near Field Communication
NTNU	Norwegian University of Science and Technology
O&M	Observations and Measurements
OGC	Open Geospatial Consortium
ONS	Object Name Service
RFID	Radio Frequency Identification
SensorML	Sensor Model Language
SOS	Sensor Observation Service
SPS	Sensor Planning Service
SWE	Sensor Web Enablement
TML	Transducer Markup Language
UHF	Ultra-High Frequency
UML	Unified Modeling Language
XML	Extensible Markup Language

B INTRODUCTION TO EPC INFORMATION SERVICES

EPCIS is the upper layers of the EPCglobal Architecture Framework. This standard is about sharing data, and specifically sharing EPC-related data. EPCIS is all about events, and the data shared is events occurring in a business process.

EPCglobal defines the goal of EPCIS as the following: *"The goal of EPCIS is to enable disparate applications to leverage Electronic Product Code (EPC) data via EPC-related data sharing, both within and across enterprises. Ultimately, this sharing is aimed at enabling participants in the EPCglobal Network to gain a shared view of the disposition of EPC-bearing objects within a relevant business context." (EPCglobal 2007a)*

This means that EPCIS is about sharing data, both within and across enterprises, and is at a higher level than the other standards in the framework which are about the data itself and how to gather it.

An important difference between EPCIS and the lower layers of the framework is that EPCIS deals with historical data in addition to real-time data. Another major difference is that EPCIS is at a semantically higher level than the lower layer because of its context-awareness. An EPCIS-level event consists not only of EPC data, but it also sets this data in a business context, saying for instance where in a business process the data was gathered.

This standard takes a service approach by defining standard interfaces for capturing and querying data through a set of service operations. It also defines a set of data standards for the shared data. The information shared can be described as the *what, where, when and why* of events (Asher, Morgan et al. 2007). This may be information such as *what* goods passed through a point, *where* that point is located, *when* it passed through and *why* it passed through. The *why* may be several different reasons, such as for example the business transaction the goods are a part of, what business step in a process is performed or the disposition of the goods. The main point is that the information shared is about events, both historical and real-time events. We will later go into more detail on the different event types available in EPCIS.

The requirements for the EPCIS standards are much more complex than for the lower level standards of the framework, due to historical data and incorporation of business process context. Thus the EPCIS standard is described as a framework specification with more detailed specifications to populate the framework.

B.1 ARCHITECTURE

EPCIS is as mentioned the upper layers of the EPCglobal Architecture Framework, and the entire framework is shown in Figure 34. We have emphasized the figure by dividing it into three different parts. This is not part of the standard, but we believe there is a distinct difference between these three, and it also helps define the parts that will be discussed in this report. At the bottom of the figure there are a set of standards for how RFID tags are to be encoded, the specification of EPC, interfaces for RFID readers and an interface for applications that filters and collects the information retrieved by the readers. These standards are mostly RFID specific, and we will not say much about them. In the middle there is a set of interfaces for capturing, storing and accessing this information, which are the EPC Information Services. This is the main focus of our project. At the top there is a set of core services that are either operated by the EPCglobal or EPCglobal has granted access to other organizations or companies to operate. These services are freely available for anyone using the EPCglobal network, and we will describe the ones related to EPCIS later in this chapter.

The architecture consists of roles and interfaces where the roles are thought implementations that are not a part of the standard, only the interfaces are defined. For EPCIS this means that the relevant parts are the EPCIS Capture Interface and the EPCIS Query Interface. This means that for instance an EPCIS Accessing Application may be implemented in a number of different ways; the only requirement is that it is able to communicate with the EPCIS Query Interface. Other interfaces will come, and we can see that the Subscriber Authentication and EPCIS Discovery roles communicate with interfaces that are TBD, meaning that they are not yet defined in the standard. These interfaces will probably become a part of the EPCIS specification in a future version. We will talk more about the EPCIS Discovery interface later, as this is important in a supply chain with multiple, and possibly unknown, parts.

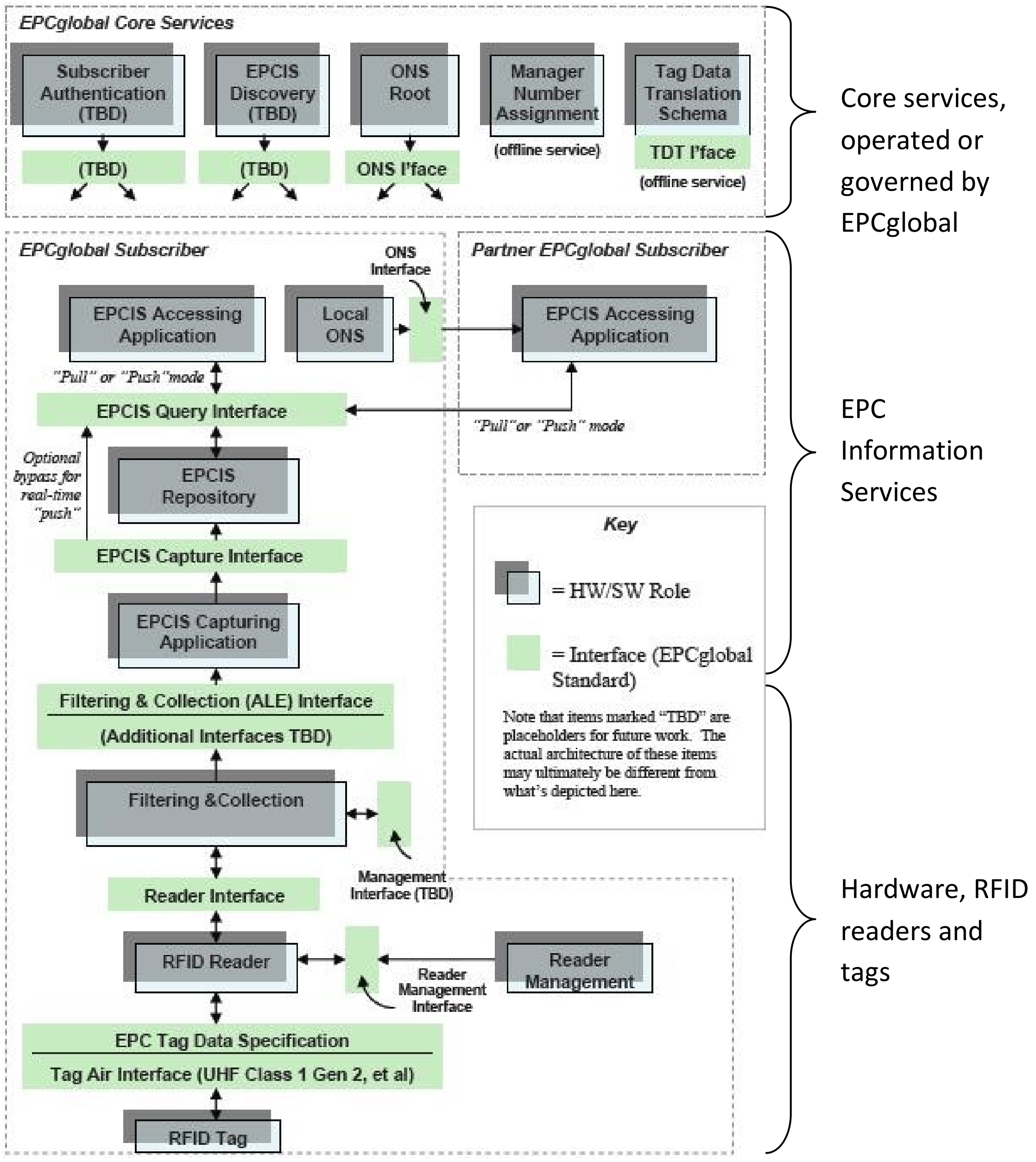

EPCglobal Core Services
Subscriber Authentication (TBD)
EPCIS Discovery (TBD)
ONS Root
Manager Number Assignment
Tag Data Translation Schema
(TBD)
(TBD)
ONS I'face
(offline service)
TDT I'face
(offline service)
EPCglobal Subscriber
ONS Interface
EPCIS Accessing Application
Local ONS
"Pull" or "Push" mode
EPCIS Query Interface
Optional bypass for real-time "push"
EPCIS Repository
EPCIS Capture Interface
EPCIS Capturing Application
Filtering & Collection (ALE) Interface
(Additional Interfaces TBD)
Filtering &Collection
Management Interface (TBD)
Reader Interface
RFID Reader
Reader Management Interface
Reader Management
EPC Tag Data Specification
Tag Air Interface (UHF Class 1 Gen 2, et al)
RFID Tag
Partner EPCglobal Subscriber
EPCIS Accessing Application
"Pull" or "Push" mode
Key
= HW/SW Role
= Interface (EPCglobal Standard)
Note that items marked "TBD" are placeholders for future work. The actual architecture of these items may ultimately be different from what's depicted here.
Core services, operated or governed by EPCglobal
EPC Information Services
Hardware, RFID readers and tags

Figure 34 EPCglobal Architecture Framework (EPCglobal 2007c)

There are three important principles that have been in mind when designing the EPCIS specification. The specification is, as mentioned before, described as a framework specification with more detailed specifications to populate the framework, and it is designed to be *layered*, *extensible* and *modular*.

B.2 ARCHITECTURE LAYERS

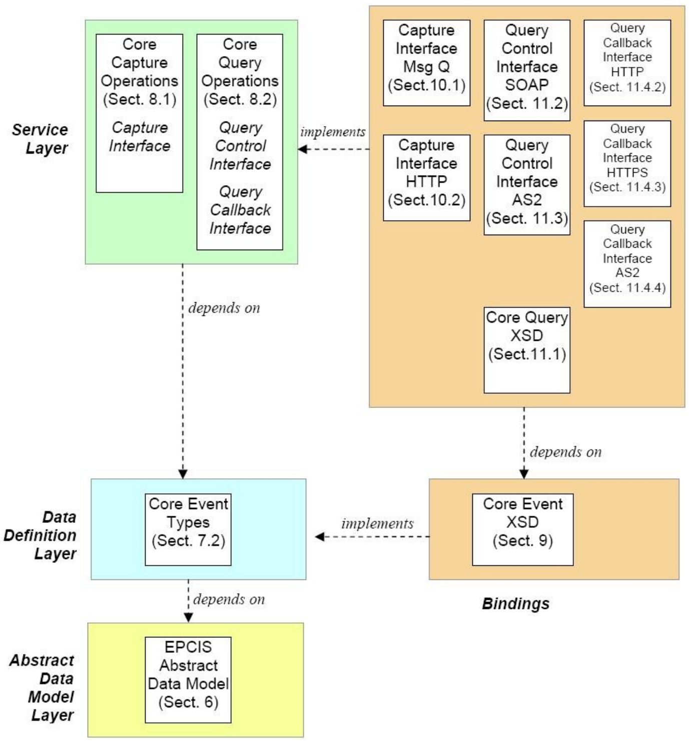

Service Layer
Core Capture Operations (Sect. 8.1)
Capture Interface
Core Query Operations (Sect. 8.2)
Query Control Interface
Query Callback Interface
implements
Capture Interface Msg Q (Sect.10.1)
Query Control Interface SOAP (Sect. 11.2)
Query Callback Interface HTTP (Sect. 11.4.2)
Capture Interface HTTP (Sect.10.2)
Query Control Interface AS2 (Sect. 11.3)
Query Callback Interface HTTPS (Sect. 11.4.3)
Query Callback Interface AS2 (Sect. 11.4.4)
Core Query XSD (Sect.11.1)
depends on
depends on
Data Definition Layer
Core Event Types (Sect. 7.2)
implements
Core Event XSD (Sect. 9)
Bindings
depends on
Abstract Data Model Layer
EPCIS Abstract Data Model (Sect. 6)

Figure 35 EPCIS layers (EPCglobal 2007a)

EPCIS consists of four different layers as shown in Figure 35. These are the Abstract Data Model Layer, Data Definition Layer, Service Layer and Bindings. The Abstract Data Model Layer specifies the structure of EPCIS data in a generic way, and is the only layer that is not extensible. The Data Definition Layer specifies the data that is exchanged through EPCIS following the rules of the Abstract Data Model Layer. The Service Layer specifies service interfaces that EPCIS clients use to interact. Bindings specify concrete realizations of the Service Layer and the Data Definition Layer. We will here give an overview of the layers and look into detail on some important parts. For detailed information we refer to the EPCIS standard.

B.3 ABSTRACT DATA MODEL LAYER

The Abstract Data Model Layer defines two kinds of data, event data and master data. This is the only layer in the specification that is not extensible; the only way to change this layer would be to change the EPCIS specification itself.

Event data comes from business processes, or rather it appears during the course of a business process, and is captured by the EPCIS Capture Interface. The event data is made available through the EPCIS Query Interface.

Master data provides context to the event data and is available through the EPCIS Query Control Interface. There is currently no interface to capture or add master data to the system, but this is expected to be added in a future version of the standard and it is not a big issue. As master data may change over time, these changes are expected to be infrequent compared to the rate event data is generated. It is however important to define the structure of the master data, as this will be shared between different parts in a supply chain.

The difference between event data and master data is that while the event data describes events on the form of *what*, *where*, *when* and *why* as described above, the master data gives more meaning to the event data. An example on event data may be "EPC 123 was at 10:30 on November the 1st 2007 observed at Location A." This does not give much meaning without knowing what EPC 123 and Location A is, and this is described by the master data.

The standard defines a set of terms for the structure of event and master data, such as Event Data, Event, Event Type, and so on. We will not go into detail on these here, for more information we refer to (EPCglobal 2007a). One term we will say something about is *Vocabulary*. A vocabulary is a finite set of alternatives that may appear in specific fields of events. There are two main kinds of vocabularies, *Standard Vocabulary* and *User Vocabulary*. A standard vocabulary is defined by organizations of many end users, such as EPCglobal, and the corresponding master data is defined by the same organizations. Changes to such vocabularies are usually done through a deliberate process, such as agreeing on a new version of a standard or a vote in an industrial group. A user vocabulary on the other hand is under the control of a single organization. The master data associated with such vocabularies are usually distributed through the EPCIS Query Control Interface. The EPCIS specification does not distinguish between these two types of vocabularies, but it is useful to think of them as two different types due to the way they are defined and modified.

We mentioned above that this is the only layer that is not extendible. By this we mean that there is only one data model, the EPCIS Abstract Data Model as shown in Figure 35, and it is not possible to extend this. It is however possible to extend parts within the data model, such as adding new vocabulary elements.

B.4 DATA DEFINITION LAYER

This layer builds on the Abstract Data Model Layer, and modules here populate the module in the Abstract Data Model Layer. The standard specifies a set of rules for how modules in this layer are to be developed, and then it presents one such module, the Core Events Module. This layer is open for extension by writing modules that extend the Core Events Module.

A module in the Data Definition Layer has the following components:

- Value types

- Event types
- Event fields
- Vocabulary types
- Master data attributes
- Vocabulary elements

Data Definition Layer modules in specifications from the EPCIS Working Group will include definitions of the first four components, but not master data attributes or vocabulary elements. Modules defined by others will include event fields extending existing event types, master data attributes extending vocabulary types and vocabulary elements to populate existing vocabularies.

For each event there is a set of assertions that defines the semantics of the event. These are either retrospective or prospective. Retrospective semantics tell us something that was known to be true when the event happened, such as item A passed gate 1. Prospective events tell us something about the present condition of the object, about what is true until contradicted by a subsequent event. For instance, if door 1 leads to room B, an object passing in door 1 should be in room B. However, from an EPCIS Accessing Application this should be considered as something that might be true, it is not a definite truth. If for instance the reader at door 2 is broken, the object may have passed through that door without the event being registered.

A restricted form of UML is used to specify event types and event fields. The UML used consists of classes with attributes and associations, but without operations. Every Event Type must include an extension point, written as <<extension point>> in UML, and Value Types may include such extension points. An extension point is a place where the implementation must allow extensibility by adding new data members. This again shows the emphasis on extensibility in the standard, extension is not an option, it is mandatory.

Core Event Types Module

This module specifies the EPCIS data capture events. The components of this module are value types, event types, event fields, vocabulary types, master data attributes and vocabulary elements. There is a set of data types that may be used for the different components in the module. We will here give a brief description of the more important ones, before we describe the different event types.

PRIMITIVE DATA TYPES

The primitive data types in this module are int, time and EPC. We mention this because it is important to know that en EPC may be both a primitive data type and a vocabulary element. This is due to the diversity of the EPC; an EPC describing a specific trade item is not considered to have master data and is represented as a primitive type, while an EPC representing for instance a read point in an ObjectEvent is a vocabulary element.

ACTION TYPES

Another important type is the Action type, which says something about how the event relates to the lifecycle of the entity described. Action type is enumerated and has three possible values: ADD, OBSERVE and DELETE. What is special with this type is that the values have different meanings for the different event types. We will go into more details about this in the section about events, but for example the action ADD for an AggregationEvent means that the EPCs have been added to a parent EPC, while for the TransactionEvent it means that it has been associated to the business transaction in question.

LOCATION TYPES

Another type we want to mention is the Location type. Four types are defined, and the special part here is that only two of them are actual location types in respect to the rest of EPCIS. The other two are only included to say that these kinds of locations are not real EPCIS locations in order to avoid confusion. This is done because there is a distinct difference between an EPCIS location and a reader identity, but these have been mixed in the past. The reader/location types, which are not true EPCIS location types, are PhysicalReaderID and LogicalReaderID. The first is the serialized identity of a reader while the second is the name given to such a reader. More information on these locations as they appear in lower layers of the EPCglobal Architecture Framework, where they are an actual part of the specification, may be found in (EPCglobal 2005) (EPCglobal 2008a). The other two location types, which are true EPCIS location types, are ReadPointID and BusinessLocationID. The first refers to the point where the event took place, and the second refers to the place where the object is after the event. For example if the reader is located at a door leading to a storage room, the ReadPointID will be referring to the door and the BusinessLocationID will be referring to the storage room. If we relate this to the two types of semantics described above, ReadPointID relates to retrospective semantics while BusinessLocationID relates to prospective semantics.

OTHER TYPES

The other data types defined are Business Step, Disposition, Business Transaction and EPCClass. Business step denotes a step in a business process, for example “sending”, disposition denotes the business state of an object, for example “in storage”. Business transaction consists of an optional id for the transaction type and a BusinessTransactionID. This is a user vocabulary, and may for instance contain the element “Corporation order #34”. The last type is the EPCClass which is a user vocabulary that denotes classes of trade items. This relates to the object field of the EPC code. Master Data Attributes for these classes is independent of EPCIS and contains the data defined for the objects, for example data from a product catalog.

EPCIS EVENTS

Five different event types are defined, one generic, EPCISEvent, and four subclasses: ObjectEvent, AggregationEvent, QuantityEvent and TransactionEvent. An overview of the different events is shown in Figure 36.

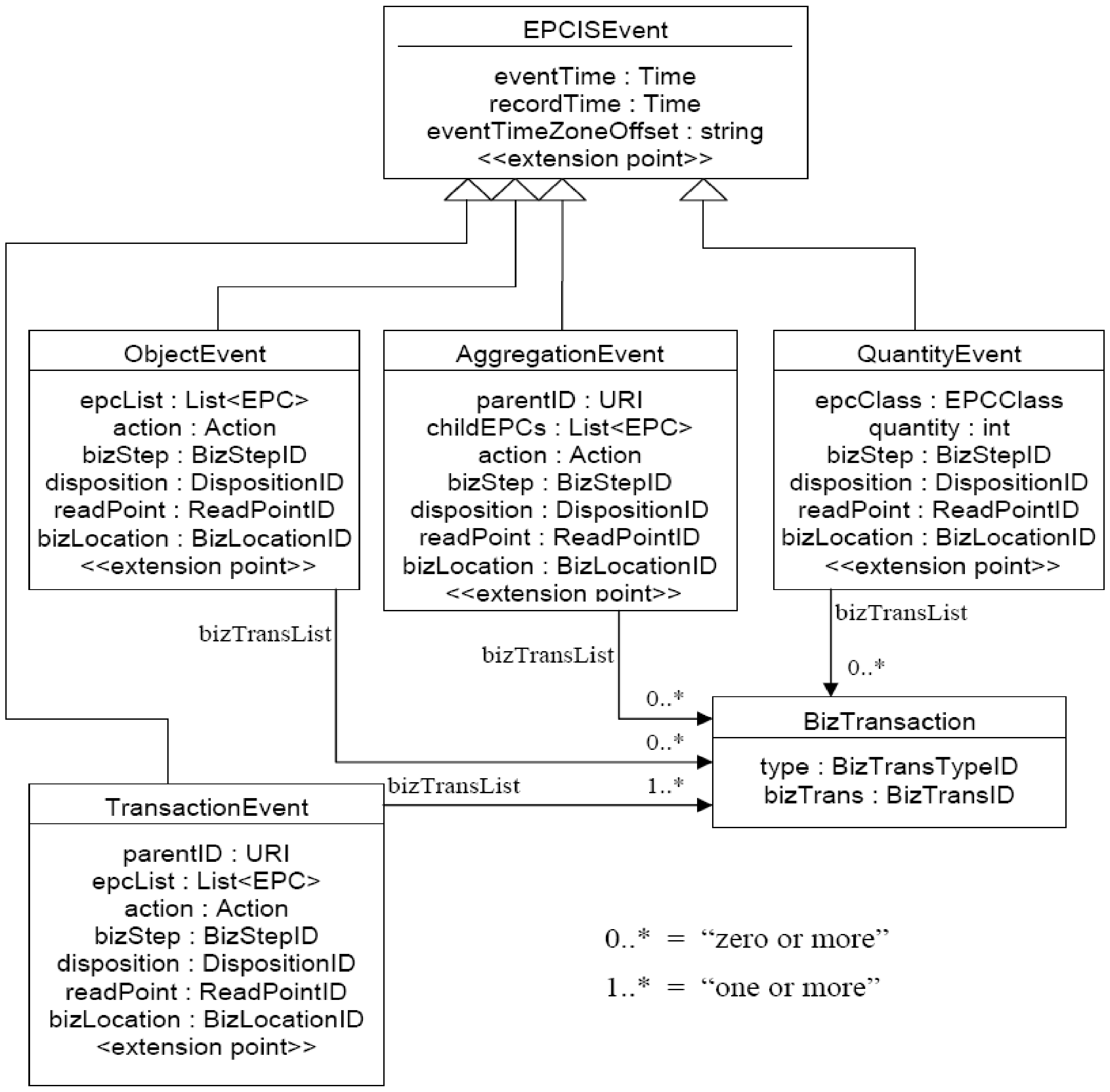


Figure 36 EPCIS event types (EPCglobal 2007a)

The EPCISEvent has only three fields, eventTime, recordTime and eventTimeZoneOffset. For each of the other event types there are four key dimensions represented by fields. These are the objects or entities of the event, the date and time, location of the event and business context. It is possible for different industries to define specifications on top of this one with industry-specific information fields.

OBJECTEVENT

This event class contains information on one or more objects identified by EPCs. If there are more than one EPC involved in the event, no relationships between them are implied. This is the most basic of the four events, and the action type only says if an EPC has been added to an object, observed or removed from an object.

AGGREGATIONEVENT

Aggregation events are about objects that have been physically aggregated. A containing entity with a set of contained elements is defined. This event type is for the cases where there is a strong relationship between the objects, for example cases that are loaded onto a pallet, where the pallet is the containing object containing the cases. It is important that this is intended for strong relationships. For instance if there are two pallets on the same truck, but not physically paired together, an ObjectEvent would be more appropriate.

QUANTITYEVENT

Quantity events are as the name implies about the quantity of an object class. This is useful for inventory levels of a product. This is the only event that does not contain one or more EPCs, but it contains EPCClass and the quantity of the class.

TRANSACTIONEVENT

This event type is used to associate one or more business transactions with one or more objects. When using the action type observe, this event type may be used to locate a set of EPCs based on transaction id instead of EPC. The transaction id field is specified such that it is possible to use internal order or transaction numbers and still having a unique id when searching for events related to a specific transaction.

B.5 SERVICE LAYER

The Service layer defines the interfaces that need to be implemented and used by the different EPCIS applications. There are three different interfaces in this layer, EPCIS Capture Interface, EPCIS Query Callback Interface and EPCIS Query Control Interface. Figure 37 show the association between the different interfaces. If we relate this to Figure 35 we can see that the capture interface is a part of the Core Capture Operations Module, and the query callback and query control modules are parts of the Core Query Operations Module. We will not go into detail on these modules, but will give an introduction with focus on the different interfaces.

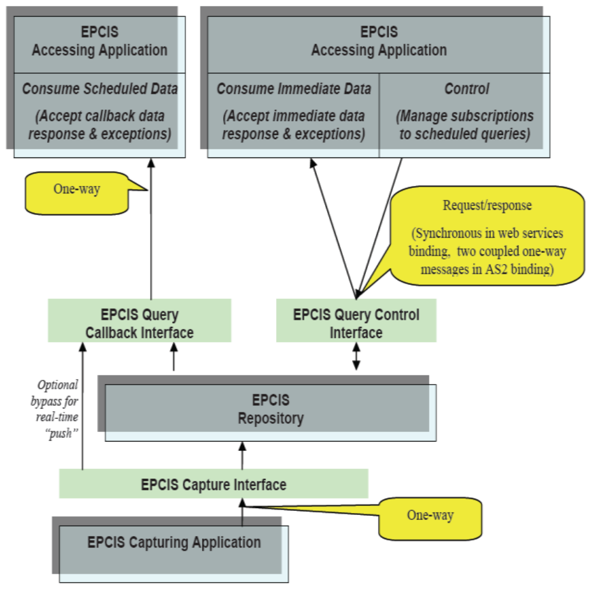

EPCIS Accessing Application
Consume Scheduled Data (Accept callback data response & exceptions)
EPCIS Accessing Application
Consume Immediate Data (Accept immediate data response & exceptions)
Control (Manage subscriptions to scheduled queries)
One-way
Request/response (Synchronous in web services binding, two coupled one-way messages in AS2 binding)
EPCIS Query Callback Interface
EPCIS Query Control Interface
Optional bypass for real-time "push"
EPCIS Repository
EPCIS Capture Interface
One-way
EPCIS Capturing Application

Figure 37 Service Layer interfaces (EPCglobal 2007a)

Core Capture Operations Module

This is a very small module; it contains only one interface, the capture interface, and is only used to deliver events from an EPCIS Capture application. This interface contains only one method, capture, and it takes a list of EPCISEvent objects as argument and has no return type.

Core Query Operations Module

This module provides interfaces for querying EPCIS data by EPCIS Accessing Applications. Two interfaces are defined, the Query Callback Interface and Query Control Interface. The module supports on-demand queries, where the result is directly returned to the client via the Query Control Interface in response to a query, and standing queries where a client registers for a query and results are returned via the Query Callback Interface. Queries for Master Data may only use the on-demand type. For the standing queries there are two possible ways for the data to be returned. The first is on a defined schedule, where results are sent on even time intervals, and the second is by specifying a trigger event that will trigger the execution of the query.

A set of exception types are defined to handle error conditions. This is important as it is possible to create requests for a very large amount of data, or requests that take too much system resources to process. Therefore there are specific exceptions defined for these cases, called QueryTooLargeException and QueryTooComplexException.

QUERY CONTROL INTERFACE

The Query Control interface is used both for on-demand queries and for administration of the standing queries. This is done with three methods: *subscribe* and *unsubscribe* for standing queries and *poll* for on-demand queries. It also contains four methods to retrieve basic information about the query service: *getQueryNames*, *getSubscriptionIDs*, *getStandardVersion* and *getVendorVersion*.

Currently there is no way for a client to specify new queries, and so the only queries available are pre-defined. In EPCIS 1.0 there are two pre-defined queries that must be implemented, but the organization creating the service may implement as many queries as it wants. The two mandatory queries are SimpleEventQuery and SimpleMasterDataQuery. Although only two, these are heavily parameterized, and it is possible to define precisely the data of interest. The first one is available both for poll and subscribe, but the latter only for poll. This is a general rule, queries for master data is only available on-demand, not for subscription, because master data is assumed to be almost static.

QUERY CALLBACK INTERFACE

The Query Callback Interface is used to return the results of standing queries to an EPCIS Accessing Application. This interface contains three methods, one for returning the results and two for the exceptions QueryTooLargeException and ImplementationException. As the response from this interface does not come directly from a request, it is necessary to use these two methods to send exceptions as there is no method called that may throw them.

B.6 BINDINGS

Bindings are defined for each of the modules in the Data Definition and Service layers. Bindings are concrete realizations of the different modules, and there may be more than one binding for each module. EPCIS 1.0 has a total of nine bindings. The Data Definition module has two XML bindings, one for event data and one for master data. The Core Capture Operations module has two bindings; one for Message Queue and one for HTTP. The rest of the bindings are in the Core Query Operations module. The Query Control interface has bindings to SOAP/HTTP and AS2 and the Query Callback interface has bindings to HTTP, HTTPS and AS2.

We will not describe the different bindings here, as these are available from the standard itself. We want to emphasize the use of standard protocols for the bindings, such as HTTP and SOAP. The exception is Message Queue, which has no industry standard. Thus the Message Queue binding is designed to apply to any system, and is more like guidelines for how to implement a binding, whereas for instance the XML bindings for data types are actual XML Schemas.

B.7 EXTENSIBILITY

Extensibility is important for the EPCIS standard, and it is reflected in just about every part of the standard. The layered approach makes it possible to change parts of the specification without having to revise the entire specification. It is for example possible to create many different bindings, and as such change the Bindings-layer, and still use the rest of the standard as it is. There are also other techniques used to enable this extensibility, for example subclassing. When subclassing a data definition in the Data Definition layer it is possible to add new functionality or information to a class, while still being able to work together with users that only know about the super class. This ensures backwards compatibility, but users without the new class will of course not be able to use the new functionality.

Another important technique is extension points. Every data definition and interface includes extension points, making it possible to extend the functionality without using subclasses. The standard is very clear on where such extension points are required, optional or forbidden. This ensures a good indication on where to add wanted functionality, and where to look for possible extensions.

It is easy to see why there is such a focus on extensibility in this standard. For a global standard to be useful it is very important that it is accepted and taken into use by as many as possible. This standard is intended to work across many different industries, and they can have very different needs. Thus it is important for these industries to extend the specifications where needed so that it applies to their specific needs. This is also the reason why the extension points try to promote both backwards and forwards compatibility, meaning that old version is compatible with an extended version and new extend version is compatible with older versions. It is also possible to have extensions that are non standard, in order to meet the specific needs of different industries and also as a possibility to experiment in order to expand and update the standard specification at a later point (EPCglobal 2007c).

B.8 MODULARITY

The last design principle, in addition to layers and extensibility, is modularity. This is a way to say that the standard is not one large specification, but as we have already mentioned it is a framework with many individual specifications that populate the framework. This enables the standard to evolve and change without having to change the entire specification; one only has to update the module that needs to be changed. Modularity is achieved through the other two principles and by explicitly grouping the functionality into modules.

While the standard is specified in a modular fashion, this does not necessarily mean that an implementation needs to be modular. It is for instance possible to combine different modules in one service, such as bindings, data definition modules and one or more services, and deploy this as one service with no trace of the modules from the standard, not externally and not in the source code. This is allowed as long as the implementation conforms to the standard. This is another example of the flexibility of the standard.

B.9 INTERFACES VERSUS IMPLEMENTATIONS

The entire specification defines interfaces and not implementations. This is done in order to create the possibility for different vendors to create their own solutions in a competitive market, but still being able to interoperate. It will also be possible for a company that wants to implement this to choose hardware and software from different vendors, and use the different products together. This is important because it means that a choice taken on for instance RFID readers will not have an impact on how the EPCIS system needs to be implemented. The only requirement for each part of a system-architecture is that it conforms to the standard specification and then they will be able to interoperate.

It also means that an implementer is not locked to for instance a specific program language or operating system. This again shows that EPCglobal tries to get a wide adaptation of the standard, and that each user is free to choose how he wants his system to be implemented.

B.10 WHAT IS MISSING?

Even if this is a very large standard which has gone through a lot of reviews, it still has some shortcomings. At the time of this writing the standard is only in its first version 1.0 (or 1.0.1 but the difference between those is just errata corrections) and not every part of the standard is finished. We will here discuss two of the features we miss the most, EPCIS Discovery and centralized authentication. Both of these are mentioned in the standard, but are marked as under development or to be developed.

EPCIS Discovery

The functionality of the EPCIS Discovery interface is not yet defined and only thoughts and envisioned requirements are mentioned in the standard (EPCglobal 2007c). This means that it may change when the final specifications are ready, but the core functionality of what it will provide is assumed to be the same.

When EPC and EPCIS become widely used and adopted, there will be a very large number of EPCIS services available. In a supply chain with many different participants, often not known in advance, it can be very difficult to locate the needed services. If someone for instance has a product they want to trace back to the manufacturer, they might not know in advance which transporting company delivered the product; it might even have passed through different companies on the way. In such a situation it will be almost impossible to locate the relevant EPCIS services, they would have to know about every single one that might be relevant and check for each product. As the number of services is expected to grow over time, this is not possible. This is the problem that EPCIS Discovery is intended to solve.

EPCIS Discovery is a service to locate EPCIS services. Conceptually it will be possible to look up an EPC and retrieve all EPCIS services with information about the given EPC. This might be combined with authorization policies, so that only services that the user has access to will be retrieved. This is listed as a Core Service in the current specification, but it is explicitly stated that this is only a placeholder, and might change in the future. It might be achieved through a set of core services, and also by services operated by EPCglobal subscribers and not only the EPCglobal itself. Regardless of how it will be implemented, this is a service that needs to be available to all users of the EPC network, and it cannot be implemented individually by every organization that wants to start using EPCIS.

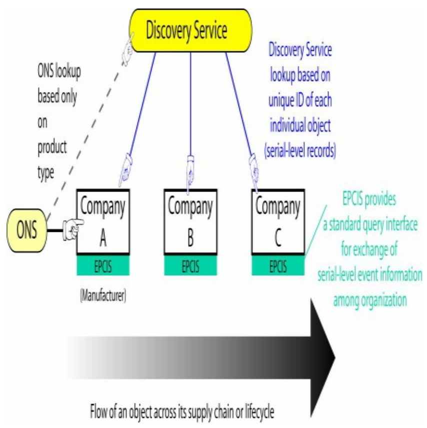

Discovery Service
ONS lookup based only on product type
Discovery Service lookup based on unique ID of each individual object (serial-level records)
ONS
Company A
EPCIS
(Manufacturer)
Company B
EPCIS
Company C
EPCIS
EPCIS provides a standard query interface for exchange of serial-level event information among organization
Flow of an object across its supply chain or lifecycle

Figure 38 Comparison of ONS and EPCIS Discovery (University of Cambridge, AT4 wireless et al. 2007)

Figure 38 shows how ONS and EPCIS Discovery complement each other. Where ONS contains product information through a link to the manufacturer of the product, EPCIS Discovery contains links to each EPCIS service with information about a given object. The lookup is also different, as ONS is based on object lookup on a class level while EPCIS Discovery is based on lookup on individual serial numbers from the EPC.

Because of the need for such a service, and because organizations and industries are already starting to use EPCIS in production environments, different industries has started to implement their own EPCIS Discovery services (Myhren 2007). If an industry or a group of trading partners is able to coordinate their efforts in creating such a service, this is a solution that is not very difficult to implement. It is more problematic for smaller companies as the development and integration costs can be too high. A solution to this is suggested in the Product Trace Service Platform (Yingping, Dong et al. 2007). Their solution is to create a discovery service and include a link to it in the ONS. This means that each EPC has two records in the ONS, one is the standard reference to the EPC Manager for that particular EPC, and the other is for the PTSP service. With this solution an application only needs to check the ONS to locate the discovery service, and all relevant services needs to be registered there. Services are registered to the PTSP service whenever an EPCIS event is registered. The company that registers an event about the EPC locates the PTSP of that EPC from the ONS and then sends a register event to the PTSP with the EPC and its own EPCIS URI. This platform is meant as a lightweight platform for prototyping or for small companies. There also exist solutions aimed at large corporations, for example from the BRIDGE project, which is funded by the European Union (University of Cambridge, AT4 wireless et al. 2007).

This and other solutions work well within an industry, but it hinders a global adaptation across different industries, as one would need to access many different EPCIS Discovery services to retrieve the information. Therefore we believe that it is important for EPCglobal to address this issue. This is a big problem for small industries or supply chains as these might not have the resources to implement their own discovery services, but for larger industries or supply chains the creation of a discovery service is not very difficult. However we believe that the lack of a common discovery service should not prevent EPCIS from being used, and (Yingping, Dong et al. 2007) shows that it is possible even for small companies to create its own discovery services. But it will make global interoperability harder and it will also make it harder to start using EPCIS than if a common service were in place.

Authentication

As we have mentioned before the specifications aim to gain a wide global adaptation as soon as possible, as a network such as this, is of no use without many participants. This is also visible with authentication; the architecture allows the use of many different authentication methods. EPCglobal does however expect that the X.509 authentication will be widely used. But a common authentication service is lacking from the current specifications. Just as for the EPCIS Discovery service a common Subscriber Authentication is mentioned in the standard, but it is marked as to be developed. We believe that this is a major problem in the current version of the standard.

Authentication is the process of identifying who is accessing a system. Authentication is important when sharing business information on the Internet; without the knowledge of who is accessing the information it is not very likely that companies would like to share this information. Therefore authentication is necessary for EPCIS. Authentication in itself is not very difficult to manage, but if every single EPCIS service provider should implement its own authentication mechanism, it would be very difficult to use. If for instance five different EPCIS services has information about a single EPC, a user that wants to access information about that EPC would have to be authenticated five times. If a company needs information on many different EPCs, this is not a good solution. Therefore it is necessary to have a common authentication service for EPCIS. Each EPCIS provider would still be able to have its own authorization policies, deciding who should have access to what, but when a request arrives it will already know its origin. That way a user will only need to authenticate once, and he will be able to retrieve the information he has access to.

It is of course possible to create shared authentication services outside of EPCglobal in the same way as for the discovery service, and this is also done by different organizations and industries today (Myhren 2007). However, security is very important to get right, and not every company would trust anyone to manage such a service. Therefore we believe that EPCglobal should prioritize to create a core service that handles authentication.