

Department of Informatics  
University of Fribourg, Switzerland  
<http://diuf.unifr.ch/>

# Radio Frequency IDentification: Evaluation of the Technology Supporting the Development of an Assets Tracking Application

Bachelor Thesis

Dominique Guinard  
September 2005



under the supervision of:

Prof. Dr. J. PASQUIER-ROCHA,  
Dr. P. FUHRER  
Software Engineering Group

Dr. O. LIECHTI  
Sun Microsystems Inc.



“Any sufficiently advanced technology is indistinguishable from magic...”

- *Arthur C. Clarke, 1961*

---

## Abstract

More than any other science, informatics jumps from a hot-topic to another. Some will die early, some will remain longer. During the last years, the increasing use of Radio Frequency as a means of identifying objects, placed this technology close to the latter category. According to many experts, the RFID (or Radio Frequency IDentification) is in the process of becoming one of our everyday-life partners.

In this fast evolving context the first contribution of this Bachelor thesis is to provide an overview of the field's "state of the art". Starting with a summary of the historical Auto-ID technologies the first part exposes the technical details and standards of the "Internet of Things": a vision shared by many computer scientists where the objects surrounding us would all be part of a global infrastructure: the EPC (Electronic Product Code) Network.

The second contribution of the present Bachelor thesis is to demonstrate a concrete use of the RFID and EPC technologies. To achieve this goal, a concrete Assets Tracking application (the RFIDLocator) was programmed on top of the Sun Java System RFID Software. The developed multi-tier application based on J2EE and the Enterprise Java Beans framework allows for tracking and locating physical objects within a set of buildings.

To test the application in real conditions the RFIDLocator was deployed on the typical hardware (i.e. RFID readers, RFID tags and middleware servers) involved in an industrial application. The description of these devices is the last contribution of this work.

**Remark:** This project has been proposed in collaboration with Sun Microsystems.

**Keywords:**

Auto-ID, Enterprise Java Beans, Electronic Product Code (EPC), EPCglobal, EPC Network, Event Manager, Internet of Things, J2EE, Java Message Service, Object Name Service, Physical Markup Language, Radio Frequency IDentification (RFID), RFIDLocator, Sun Java System RFID Software.

# Contents

<b>1. Introduction</b>	<b>1</b>
1.1. Motivation and Goals	1
1.1.1. General Study of the RFID Domain	1
1.1.2. Testing The Sun Java System RFID Software	1
1.1.3. Developing a Sample Application	1
1.2. Organization	2
1.3. Notations and Conventions	2
<b>I. Introduction to the RFID Domain</b>	<b>4</b>
<b>2. Understanding RFID</b>	<b>5</b>
2.1. Summary of Common Auto-ID Systems	5
2.1.1. Barcodes	5
2.1.2. Optical Character Recognition (OCR)	7
2.1.3. Smart cards	7
2.2. Radio Frequency Identification	8
2.2.1. Architecture of an RFID system	8
2.2.2. The RFID Transponders and RFID chips	10
2.2.3. RFID Against other Auto-ID Technologies	11
<b>3. Standards Around the EPC</b>	<b>14</b>
3.1. From the Auto-ID Center to the EPCglobal...	14
3.2. The Electronic Product Code	15
3.2.1. Anatomy of an EPC tag	15
3.2.2. About Tag Data Translation	20
3.3. The Physical Markup Language (PML Core)	21
3.3.1. Elements of the PML Core	22
3.4. The Savant	24
3.4.1. The Event Manager	26
3.5. The Object Name Service (ONS)	30
3.5.1. ONS and DNS	31
3.5.2. EPC to DNS query	32
3.5.3. Results of an ONS query	32
3.6. Towards an “Internet of Things”	33
<b>II. The RFIDLocator Application</b>	<b>35</b>

<b>4. Introduction to the Application</b>	<b>36</b>
4.1. Introduction to the Idea . . . . .	36
4.2. Introductory Terminology . . . . .	37
4.3. Use Cases . . . . .	39
4.3.1. Main Use cases . . . . .	39
4.3.2. Demonstration Use Cases . . . . .	44
<b>5. Software Architecture and Implementation</b>	<b>46</b>
5.1. Object Model of the RFIDLocator . . . . .	47
5.1.1. Location . . . . .	47
5.1.2. PhysicalAntenna . . . . .	47
5.1.3. LogicalAntenna . . . . .	47
5.1.4. Reader . . . . .	50
5.1.5. Action . . . . .	50
5.1.6. TraceableObject . . . . .	50
5.1.7. User . . . . .	51
5.1.8. LocatorObservation . . . . .	51
5.1.9. BufferedObservation . . . . .	52
5.2. The Services . . . . .	52
5.2.1. UserManager . . . . .	52
5.2.2. LocationManager . . . . .	53
5.2.3. TraceableObjectManager . . . . .	53
5.2.4. ReaderManager . . . . .	53
5.2.5. PMLSimulatorPublisher . . . . .	55
5.2.6. ObservationManager . . . . .	56
5.3. The SensorListenerMessageDrivenBean . . . . .	56
5.4. The Solvers . . . . .	56
5.4.1. Introduction to the Solvers . . . . .	56
5.4.2. Architecture of the Solvers . . . . .	58
5.5. Other Classes . . . . .	60
5.6. Summary of the Sequences . . . . .	60
5.6.1. Solving Sequence . . . . .	60
5.6.2. Seeking a TraceableObject . . . . .	62
5.7. Reader's Configuration Formalism . . . . .	62
<b>6. Hardware and Software Choices</b>	<b>65</b>
6.1. Software Choices . . . . .	65
6.1.1. The Implementation of the Event Manager . . . . .	65
6.1.2. Enterprise JavaBeans . . . . .	66
6.1.3. XDoclet . . . . .	67
6.1.4. Application Server Software . . . . .	67
6.2. Hardware Settings . . . . .	69
6.2.1. Readers and Connectors . . . . .	69
6.2.2. The Server of the Event Manager . . . . .	69
6.2.3. The Machine of the Application Server . . . . .	71
6.2.4. RFID Labels . . . . .	71
6.3. Summary . . . . .	73

---

<b>7. RFIDLocator: Use and Installation</b>	<b>75</b>
7.1. User Manual	75
7.1.1. The Graphical User Interface	75
7.1.2. Access to the Main Functionalities	77
7.1.3. Test Users	79
7.2. Installation Guide	82
7.2.1. Installing the Event Manager	82
7.2.2. Installing the RFID Reader	83
7.2.3. Installing the Application Server and the Database	83
7.2.4. JMS Settings	87
7.2.5. Deploying the Application	89
7.2.6. Starting the RFIDLocator	90
7.3. Extending the RFIDLocator	91
7.3.1. Required Software	91
7.3.2. Compiling the project	91
<b>8. Conclusion</b>	<b>93</b>
8.1. Possible Extensions	93
8.2. Final Thoughts about the RFID Technologies	93
<b>A. Common Acronyms</b>	<b>95</b>
<b>B. XML Schema</b>	<b>97</b>
B.1. PML Schema	97
B.1.1. PmlCore.xsd	97
B.1.2. Identifier.xsd	101
B.2. RFIDLocator Reader's Configuration Formalism	103
B.2.1. RFIDLocator.xsd	103
<b>C. CD-ROM</b>	<b>105</b>
<b>D. Website of the Project</b>	<b>108</b>
<b>E. Hardware and Software</b>	<b>110</b>
E.1. Software	110
E.2. Hardware	111
<b>F. Licenses of the Project</b>	<b>112</b>
F.1. License of this Document	112
F.2. License of the RFIDLocator and the JMSBridge	112

# List of Figures

2.1. The Auto-ID technologies . . . . .	6
2.2. EAN-8 representation of the number 2402820 . . . . .	6
2.3. General structure of a microprocessor smart card . . . . .	8
2.4. A typical reading sequence . . . . .	9
2.5. Basic structure of a transponder . . . . .	10
3.1. Logo of the Auto-ID Center . . . . .	15
3.2. The data of a standard EPC Tag . . . . .	16
3.3. The hexadecimal representation of an EPC (Type 1, GID, encoded on 96 bits) . . . . .	17
3.4. Pure entities EPC-GID URIs obtained by the Sun Java System RFID Software PMLReader . . . . .	17
3.5. Overview of the EPC's Identity Concept . . . . .	20
3.6. A simple, non-formalized, goods tracking message . . . . .	22
3.7. The sensor data model (from [FAOH03]) . . . . .	23
3.8. PML extract generated by a PML-Reader . . . . .	25
3.9. Components of the Savant . . . . .	26
3.10. A example of modules combining in the EM . . . . .	27
3.11. Implementation of a simple filter using Java (first part) (from: [Goy03]) . . . . .	28
3.12. Implementation of a simple filter using Java (second part) (from: [Goy03]) . . . . .	29
3.13. An ONS query by a client application . . . . .	31
4.1. Use cases on the Application Server (core application) . . . . .	40
4.2. Use cases on the Event Manager . . . . .	41
5.1. The Object Model of the RFIDLocator . . . . .	48
5.2. Elements of the EJB framework . . . . .	49
5.3. Class diagram of a LogicalAntenna . . . . .	49
5.4. Assigning actions for a gate . . . . .	50
5.5. A concrete example using a IN Action . . . . .	51
5.6. Class diagram of a LocatorObservation . . . . .	52
5.7. A typical Session Façade . . . . .	53
5.8. A very simple XML code . . . . .	54
5.9. The marshalling/unmarshalling process using JAXB . . . . .	54
5.10. The JAXB unmarshalling of a XML string . . . . .	55
5.11. A typical RFIDLocator reader's setting . . . . .	57
5.12. A concrete example of capturing the direction of the motion . . . . .	59
5.13. The Solvers provided with the RFIDLocator . . . . .	59
5.14. Simplified sequence diagram of an Observation solving . . . . .	61

---

5.15. Seeking a TraceableObject: simplified sequence diagram . . . . .	62
5.16. Diagram of the Reader's Configuration Formalism . . . . .	63
5.17. XML document based on the RFIDLocator_v.0.xsd schema . . . . .	64
6.1. The Event Manager and the Application Server . . . . .	66
6.2. A code extract containing XDoclet instructions . . . . .	68
6.3. Front view of the Medio L100 . . . . .	70
6.4. The Moxa DE-311 Serial to TCP/IP converter . . . . .	70
6.5. Schematic view of the server (from [Sun97]) . . . . .	71
6.6. The actual Ultra Enterprise 450 hosting the Event Manager . . . . .	72
6.7. The computer hosting the RFIDLocator . . . . .	72
6.8. Back view of a Philips Rafsec tag . . . . .	73
6.9. The hardware devices of the demonstration . . . . .	73
7.1. The thin-client GUI of the RFIDLocator . . . . .	76
7.2. Creating a new user . . . . .	77
7.3. Readers' configuration page . . . . .	78
7.4. Attach / Detach a Tag . . . . .	79
7.5. Seeking a Traceable Object . . . . .	80
7.6. The PMLSimulator . . . . .	81
7.7. Medio L100 configuration for the RFIDLocator . . . . .	84
7.8. JMS Logger configuration for the RFIDLocator . . . . .	85
7.9. Welcome screen of the Sun Java System Application Server . . . . .	86
7.10. The JMS queues of the RFIDLocator . . . . .	87
7.11. Configuring a new Connection Factory . . . . .	88
C.1. Tree view of the CD-ROM's content (first part) . . . . .	106
C.2. Tree view of the CD-ROM's content (second part) . . . . .	107
D.1. Screenshot of the project's official web-page . . . . .	109

# List of Tables

2.1.	A quick overview of the different smartcards . . . . .	7
2.2.	Most important RFID chips . . . . .	11
2.3.	The two main types of transponder's power-supply . . . . .	11
2.4.	Some of the transponder's housing available on the market . . . . .	12
2.5.	Comparison of some Auto-ID systems (from [Fin03] and [She04]) . . . . .	13
3.1.	Filters provided by Sun in the RFID software solution . . . . .	27
3.2.	Loggers available in the Sun Java System RFID Software . . . . .	30
3.3.	A typical set of NAPTR records . . . . .	33
7.1.	Test users . . . . .	80
7.2.	Physical Destinations required by the RFIDLocator . . . . .	86
7.3.	Queues and Connections Factories used by the RFIDLocator . . . . .	89
7.4.	Physical Destinations required by the RFIDLocator . . . . .	89

# Preamble

## Acknowledgment

### Special thanks...

Before beginning with the actual matter of this Bachelor Thesis, I would like to thank my two supervisors on this project: Dr. Patrik Fuhrer (from the Software Engineering Group) and Dr. Olivier Liechti (from Sun Microsystems). Their support, sincere engagement and fellowship went far beyond my expectations.

Thanks as well to the Prof. Dr. Jacques Pasquier-Rocha for giving me the opportunity to achieve my Bachelor Thesis in his group and for providing me with precious advices for this document.

A special thank to Marc Vidal-Alaiz from Sun Microsystems France for helping me solving some tricky hardware issues.

Finally the last thank goes to my family for their everyday support and love.

### About this Bachelor Thesis

All the sources of this document are available for free from the website of the project (see Appendix D). It is distributed under the GNU Free Documentation License (see Appendix F). It was composed using  $\text{\LaTeX}$ .

## Comments and questions

Comments, questions and error reports are welcome:

Dominique Guinard

Rte de Planafin 15

1723 Marly

Email: [dominique.guinard@unifr.ch](mailto:dominique.guinard@unifr.ch) or [dguinard@gmipsoft.com](mailto:dguinard@gmipsoft.com)

Personal website: [www.gmipsoft.com/unifr](http://www.gmipsoft.com/unifr)

Website of the project: [www.gmipsoft.com/rfid](http://www.gmipsoft.com/rfid) <sup>1</sup>

The Software Engineering Group at the University of Fribourg, Switzerland can be contacted at:

---

<sup>1</sup>This URL is a shortcut to the official website on the University server: <http://diuf.unifr.ch/softeng/student-projects/completed/guinard/index.html>

Prof. Jacques Pasquier-Rocha  
Department of Informatics  
University of Fribourg  
Rue de Faucigny 2  
CH-1700 Fribourg  
Website of the group: <http://diuf.unifr.ch/softeng>

# 1

## Introduction

### 1.1. Motivation and Goals

This document and the provided application (the RFIDLocator) are the result of a Bachelor Thesis at the Faculty of Science of the University of Fribourg (Switzerland). It was proposed by the Software Engineering Group (Department of Informatics) in cooperation with Sun Microsystems Inc.

The main goal of this work is the development of an application demonstrating the capabilities of the Sun Java System RFID Software and the overall RFID (Radio Frequency Identification) technologies. As the RFIDs and their massive use in the industry are still a relatively new topic, some research and documentation objectives were added to the first goals of the project.

The following subsections summarize the main objectives of the project and offer a global view of the work.

#### 1.1.1. General Study of the RFID Domain

A general study of the RFID related domain is provided. It includes a study of documents, books, articles and websites on the subject. The aim of this first part is to:

- Understand the consequences and application domains of the technology.
- Discover the main actors (tags manufacturer, software integrators and developers, technology leaders, etc.) of this market.
- Understand the place of the technical components a typical RFID architecture requires.

#### 1.1.2. Testing The Sun Java System RFID Software

A technical study of the middleware solution offered by Sun (Sun Java System RFID Software v.1.1.0, see [34]) is also provided. After studying the architecture of the system and its two key components (the Event Manager and the Information Server), a summary of the advantages it offers is provided.

#### 1.1.3. Developing a Sample Application

The last goal of this Bachelor work is to develop an application which aims to illustrate the use of the RFID technologies and the Sun Java System RFID Software.

## 1.2. Organization

The present Bachelor Thesis is structured according the main objectives of this work:

1. Part I is an introduction to the Automatic Identification (Auto-ID) technologies. It provides the reader with a summary of the existing techniques, focusing on the RFID. It also develops some important subjects related to the EPC Network, widely known as a key component of “The Internet of Things”.
2. The last objective of this work is to design and program an application demonstrating the capabilities of the Sun Java System RFID Software v.1.0. Thus, Part II of this paper also contains a report of the work achieved towards this application.
3. Eventually a brief conclusion offers some subjective remarks and outlooks regarding the RFID world.

Besides these main parts:

- Appendix A summarizes the acronyms widely used in the field of the RFID and the Auto-ID.
- Appendix B.1 offers a copy of the schema forming basis of the PML language (see Section 3.3) and of the RFIDLocator Reader’s Configuration Formalism.
- Finally Appendix C provides a CD-ROM containing the source files of the project as well as a digitalized trace of the most important documents and binaries used for this Bachelor work.<sup>1</sup>.

## 1.3. Notations and Conventions

- **Path:** is used for filenames, pathnames, hostnames, domain names, URLs, URIs, and addresses.
- *math:* is used for mathematical elements.
- **Important:** emphasizes important terms or sentences.
- **Code:** is used for instructions, objects/classes names as well as commands.
- Longer extracts of code are emphasized using the following style:

---

`This is a longer extract of code...`

---

- The term “Sun Java System RFID Software v.1.0 ” is shortened in “Sun Java System RFID Software ” throughout the rest of this paper.
- Acronyms are detailed only when first appearing. However, the most common ones are listed in Appendix A.
- The present Bachelor Thesis is divided in two parts. Parts are split into chapters. Chapters are composed sections. Where necessary, sections are further broken down into subsections.

---

<sup>1</sup>Most of the content of the CD-ROM can also be downloaded from the project’s website (see Appendix D)

- 
- Figures are numbered inside a chapter. For example, a reference to Figure  $j$  of Chapter  $i$  will be noted *Figure  $i.j$* .
  - References to items in the bibliography are written as follow: [ACM03]. The complete bibliography is located at the end of this Bachelor Thesis.
  - References to websites are written as follow: [ $i$ ] where  $i$  is a number. A list of all the referenced websites is available at the end of this Bachelor Thesis.
  - As a respect to both genders, he and she are used interchangeably in this document.

**Part I.**

# **Introduction to the RFID Domain**

# 2

## Understanding RFID

---

<b>2.1. Summary of Common Auto-ID Systems . . . . .</b>	<b>5</b>
2.1.1. Barcodes . . . . .	5
2.1.2. Optical Character Recognition (OCR) . . . . .	7
2.1.3. Smart cards . . . . .	7
<b>2.2. Radio Frequency Identification . . . . .</b>	<b>8</b>
2.2.1. Architecture of an RFID system . . . . .	8
2.2.2. The RFID Transponders and RFID chips . . . . .	10
2.2.3. RFID Against other Auto-ID Technologies . . . . .	11

---

Automatic Identification systems (also known as Auto-ID systems) are not new. However, these systems are becoming more and more popular in many business areas. In short, the term Auto-ID groups the technologies helping computer to identify objects, animals or people. Logistics services, document management offices, security services, rental companies, supply chain managers are just a few of the nowadays Auto-ID adepts.

In use since the seventies, Automatic Identification procedures were developed in order to create means of providing information about objects<sup>1</sup> in transit. In recent years, Auto-ID technologies tended to become more flexible, enabling data to be collected anywhere, anytime and without human involvement. This is where RFID (Radio Frequency IDentification) enters the game. This technology, which is often presented as the next generation of barcodes, is a response to the industries' needs for contact less, flexible and ubiquitous identification systems.

Exposing the use of general Auto-ID systems, is of great help to understand the benefits of the RFID. As a consequence, this chapter focuses on an overview of the existing Auto-ID technologies. As an introduction, Figure 2.1 offers an overview of the different Auto-ID technologies.

### 2.1. Summary of Common Auto-ID Systems

#### 2.1.1. Barcodes

June 26, 1974, Ohio, USA: the first product using barcodes, a 10-pack of Juicy Fruit chewing gum, is scanned at the check-out counter [Ada].

Thirty years later, the market of barcodes is so wide, involving so many companies, that giving it a size is quite a difficult task. However, as an entry point consider the Western Europe market

---

<sup>1</sup>Objects is to be taken in its wide meaning, ranging from non-living objects to animals or human beings.

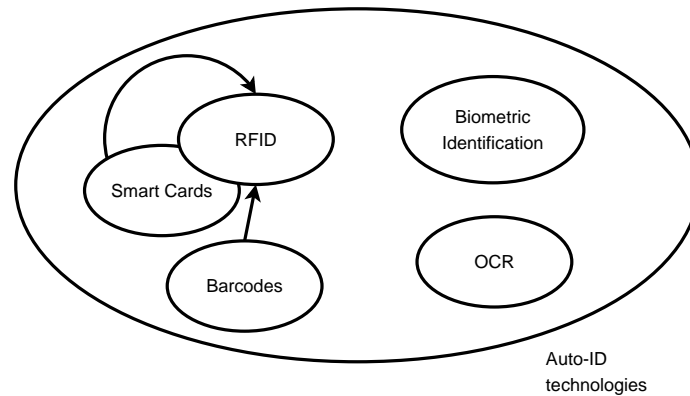


Figure 2.1.: The Auto-ID technologies



Figure 2.2.: EAN-8 representation of the number 2402820

which had a size of about DM 3 billions at the beginning of the nineties<sup>2</sup>[Fin03]!

Technically speaking, the barcode comprises a field of bars and empty spaces, vertically printed on a sticker or a product label. The sequence (bars, gaps) as well as the widths of the sticks are converted into an ASCII code (American Standard Code for Information Interchange) using optical lasers (or LEDs in cheaper systems) and a complex set of mirrors. A barcode representation of the number 2402820 is provided by Figure 2.2. Note that the trailing 2 on the figure is a checksum to control the validity of the reads.

Over the past 30 years, many barcodes standards were published, with the most popular among them certainly being the EAN (European Article Number) code. This latter is nothing but an extension of the widely used UPC (Universal Product Code) introduced in the USA in 1974.

Barcodes are a world-wide and popular mean of identifying objects. The uses of this technology range from safety applications to grocery stores. For many years, barcodes successfully held their market against the next-generation technologies. There is a simple reason for this long-lasting success: barcodes are made of paper and ink which makes these tags cheaper than any other Auto-ID system. This fact may well stay true for quite long. Indeed, however good the costs per item of the other Auto-ID systems will improve, paper and ink is likely to stay cheaper.

Understanding the way barcodes and how their standards evolved is a key issue in getting the potential of the RFID. However, since this section focuses on an overview of the subject it will not be discussed in more details here. For further reading check [She04].

<sup>2</sup>Which corresponds to about CHF 2.6 billions or € 1.6 billions.

Type	Communication	Characteristics
Memory cards	Using galvanic contact.	Cheap and widely used. Contains only a memory, usually of type EEPROM.
Microprocessor cards	Using galvanic contact.	Contains a microprocessor which facilitates rapid adaptation to new applications.
Contact-less smart cards	Without galvanic contact but need to be really close to the reader.	Can be either a memory or a microprocessor card.

Table 2.1.: A quick overview of the different smartcards

### 2.1.2. Optical Character Recognition (OCR)

Optical Character Recognition is older than barcodes and was first used in the sixties. It certainly is a well known concept for the reader as nowadays, every single home-purpose scanner is shipped with an OCR software.

This technology is basically a mean of scanning textual documents (on paper or some other physical support) to extract the information they contain. That is, not only an image of the document but a real, text-based, extraction of its content. The main difference between OCR and barcodes resides in the fact that OCR-enabled documents contain far more information than barcodes labels. The ability to read the document without human involvement is also an advantage of OCR over barcodes.

The main drawback of the Optical Character Recognition is certainly its cost and complexity when compared to other automatic identification procedures. As a consequence, OCR is mainly used for very specific applications. As an example consider the scanning of envelopes at a mail distribution center. Using Optical Character Recognition the computers are able to identify where the mail should be forwarded.

### 2.1.3. Smart cards

Invented in 1974 by Roland Moreno<sup>3</sup> and first used as prepaid phonecards in 1984, smart cards are credit-card sized pieces of plastic containing a data storage system.

To extract or write data into such a card, one needs a reader. For so called “contact” smartcards, the reader initiate a galvanic connection between the readers metallic contact pins and the card contact surface (usually a gold-plated area of about  $1\text{ cm}^2$ ). After supplying energy and clock pulse, the reader is all set to start extracting (or writing) data out of the card, at a rate between 9.6 and 115.2 kbit/s.<sup>4</sup>

According to experts, the smart cards market is one of the fastest growing sub-sectors of the microelectronics industry [Fin03]. As a consequence many types of such cards have been developed and the term “smart cards” can now be considered as a generic term for slightly different plastic-cards technologies. Table 2.1 gives an overview of the most interesting types of smart cards according to the subject of this paper.

It is worth noting at this point that the border between contact-less smart cards and RFID Tags is not very clear. Actually both technologies are overlapping and the choice of one or the other many depends on the actual use case, not much on the enclosed technology.

<sup>3</sup>Roland Moreno is a French inventor. His history of invention is a must read for anyone eager to find out about a quite atypical discoverer (see [Mor]).

<sup>4</sup>Some work is currently done to improve this speed, but at the time of writing this paper no improved standard was finalized.

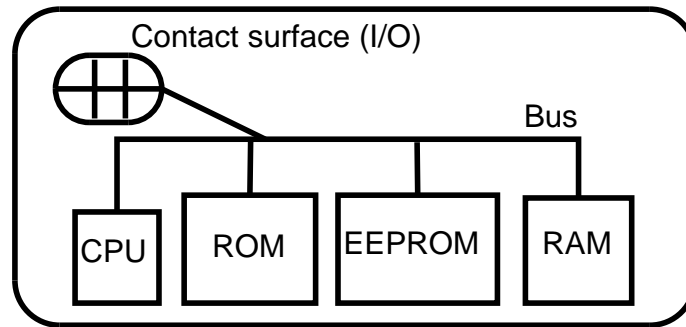


Figure 2.3.: General structure of a microprocessor smart card

Nowadays, Microprocessor smart cards have approximately the calculation power of the first IBM PC, embedded in a microchip of no more than  $0.25 \text{ cm}^2$ . Their internal organization is a quite interesting issue as it is very similar RFID tags.

A smart card has a global size of about  $40 \text{ cm}^2$ , split in:

- 98% of plastic
- 2% of electronic material

The latter part is obviously the most interesting. In a Microprocessor smart card, the microchip, is composed of five main electronic parts:

- The ROM (Read Only Memory): enclosing the operating system of the card.
- The EEPROM (Electrical Erasable Programmable Read Only Memory): is used for the permanent storage of data (like the balance of digital money in the case of an electronic purse)<sup>5</sup>.
- The RAM (Random Access Memory): transient memory keeping temporary data in memory as long as the card is powered.
- The CPU (Central Processing Unit): an 8 to 32 bits microprocessor sometimes coupled with a cryptographic coprocessor. The latter is used to perform better and faster authentication schemes (see [HMSS03]). An overview of the smart cards' general structure is provided in Figure 2.3.

Smart cards are widely used in the identification business. For instance, many company uses it to restrict access to the buildings by automatically identifying their employees.

## 2.2. Radio Frequency Identification

With their first use in tracking and access applications in the late 1980s [Wik], Auto-ID Radio Frequency transponders are basically **contactless** smart cards. Where common smart cards need a galvanic contact to be read/written, the RFID transponders can be operated using magnetic or electromagnetic fields.

### 2.2.1. Architecture of an RFID system

A common RFID system is made up of three main hardware components:

1. A **transponder**, which ranges from RFID label pasted on an object to Smart Tokens.

<sup>5</sup>The CASH card in Switzerland is a good example of electronic purse implementation.

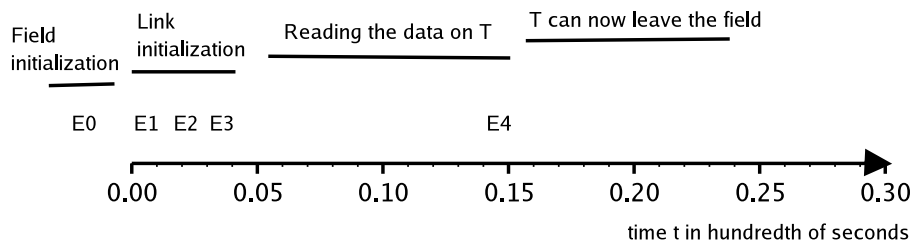


Figure 2.4.: A typical reading sequence

2. A **reader** (sometimes called “interrogator”), which is in charge of reading or writing the data contained in the transponders. It is emitting, through one or more antennae, a electromagnetic field. Depending upon the readers and the transponders which react to the field, the frequency can be either a:

- LF (Low Frequency): 125-134 KHz
- HF (High Frequency): 13.56 MHz
- UHF (Ultra High Frequency): 868 to 956 MHz
- Microwave: 2.45 GHz

Due to the transponder’s relatively low cost and to its large range of application, the HF (13.56 MHz) is currently the most widely used technology. Not far behind, the UHF reader has quite an amount of applications. In march 2005, Delta Airlines began a series of tests on UHF tags with the final objective of using them for inventory tracking. It is easy to imagine how much it can help them tracking the huge amount of parts of an aircraft<sup>6</sup>!

3. A **computer** (optional) with an RFID application, which has basically two goals:

- Processing the data sent by the reader.
- Piloting the reader.

The RFID Event Manager (see Section 3.4.1) bundled with the Sun Java System RFID Software v.1.1.0 is a good example of such an application.

The example below, gives an overview of the steps for a data transmission between a reader and a transponder. It is illustrated as well by the Figure 2.4:

Given,

- A read/write tag,  $T$ :
  - on an object in motion;
  - without power supply;
  - with a data-rate (from transponder to reader) of 79kbps.
- A reader,  $R$ .
- Two RF (Radio Frequency) antennae  $A_1, A_2$ , attached to the reader, with an activation scope of one meter each.
- A word of 10 bits to be extracted from the transponder.
- A time,  $t[100^{-1}sec]$ .

<sup>6</sup>See [Rob05] for more information about the use of UHF RFID for tracking the parts of aircraft.

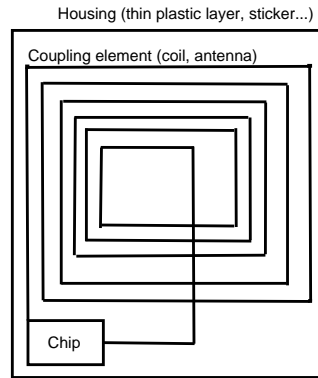


Figure 2.5.: Basic structure of a transponder

Then, the following steps have to occur for a successful data transmission:

1.  $t - 10$ :  $R$  is started and emits an electro magnetic field (High Frequency: 13.56 MHz) through  $A_1$  and  $A_2$  (Event E0).
2.  $t$ :  $T$  enters the scope of  $A_1$  (but **does not touch**  $A_1$ ), the object carrying it continues its way (Event E1).
3.  $t + 0.01$ : A special circuitry on the transponder converts the electro magnetic field to a source of power for the enclosed chips (Event E2).
4. At  $t+0.01+0.01$  the link is now active, data can be transmitted:  $R \rightarrow T$  (write operation) or  $T \rightarrow R$  (read operation). We want to transmit 10 bits ( $T \rightarrow R$ ) (Event E3) which takes 0.13 seconds.
5. At  $t + 0.02 + 0.13$ , the operation is over,  $T$  can now safely quit the scope of  $A_1$  (Event E4).

The times (delays, reading time) are provided for information only. They are based on existing tags and readers but may vary significantly depending on the exact kind of transponders, respectively readers used in the system.

Because both the scope of the antennae is bigger than just a few centimeters and the data to transmit is not really big, the above sequence can be done **in motion**. As a consequence, for many applications of the RFID technology, there is no need for the tagged object to stop moving while transmitting to the reader. This fact is already a great advantage of tags over common barcodes technologies, as readers of this latter technology are not efficient “on the move” and need a particular angle of reading.

### 2.2.2. The RFID Transponders and RFID chips

An RFID transponder (often called “RFID tag”<sup>7</sup> in the literature) is, when strictly speaking of its enclosed technology, a contactless smart card. It is basically composed of a coupling element (coil, antenna) connected to an electronic chip.

Many types of transponders exist, they mostly vary on four points:

1. The kind of embedded chip (see Table 2.2).
2. The transponder’s power-supply (see Table 2.3).

<sup>7</sup>The term “tag” is widely used to name any kind of RFID transponder. This fact can be confusing as in the early RFID literature a tag, strictly speaking, was only a subset of the overall RFID transponders.

Type of chip	Description	Example of use case	Example of chip
Read only	The data on the transponder is factory-programmed.	Animals tracking.	TAGSYS C210
Read / Write	Data can be written to the transponder using a RF reader.	Automated supply chains, aircraft engines' parts tracking.	TAGSYS C240
Programmable	Gives more flexibility about what is to be recorded in the chip.	RFID enabled car rental.	TAGSYS C220

Table 2.2.: Most important RFID chips

Type of transponder	Description	Advantages
Passive RFID	The power is supplied by the electromagnetic field generated by an antenna.	Low cost.
Active RFID	The power is supplied by a battery embedded in the transponder.	Much wider reading/writing range, enables more complex chips.

Table 2.3.: The two main types of transponder's power-supply

3. The frequency of electromagnetic field (briefly described in Subsection 2.2.1).
4. The physical properties of the housing (see Table 2.4).

An instance of an RFID transponder is basically a variation of these four main properties. The referenced figures summarize the most common configurations for each point. RFID smart labels<sup>8</sup> (see Table 2.4, fourth line), need some more attention as the application described in this paper mainly focuses on this kind of RFID transponders.

A distinction can be done between **labels** and the other housing of the transponders. Due to their housing, RFID labels are primarily a response to the market's need for a next generation of barcodes.. Other types of housing (smart tokens, smart pills, etc.) are intended to have other fields of uses.

A smart label is basically a thin sliced contactless smart card usually embedded in paper or composite material. As an example, the labels used for the demonstration application described in this paper were EPC (see Chapter 3) paper labels embedded in stickers. This make them ideal for document management or assets tracking.

Eventually, it is worth noting that a chip can be of type RFID while not EPC (Electronic Product Code) standard compliant. More information about these standards is given in Chapter 3.

### 2.2.3. RFID Against other Auto-ID Technologies

The fast, automatic and ubiquitous identification of living and non-living objects is one of the challenges of today's computer science (see [HMSS03]).

Barcodes provide a solution to identify objects for a very low per item cost, however this technology is not really flexible. First of all to identify a bar-coded object a laser-reader is required. Such a machine is a complicated set of lasers and mirrors. As a consequence, a barcode reader can only identify an object if its barcode is:

<sup>8</sup>Which we will simply call "labels" or "tags" throughout the rest of this paper.

Housing	Common name(s)	Example of use
Strong PET layer	Smart tag	Long-term tracking of outdoor objects.
Plastic token	Smart token	Smart key for automobiles.
Plastic card	Smart card, ID cards	Access control card.
Thin paper or composite foil	Smart label, smart tag	Document management, assets tracking in the supply chain.
Glass	Smart tag, smart pill	Subcutaneous injection for animals tracking.
No housing	Embedded RFID	RFID systems embedded in objects, brand protection.

Table 2.4.: Some of the transponder's housing available on the market

1. Located close to the machine;
2. In the line of sight of the reader;
3. Not moving while reading (excepted for some sophisticated and expensive readers).

This basically means object identification using such a system is not flexible enough for a pervasive and ubiquitous model. These facts could make barcode reading the weakest link of the tomorrow's (or shall we say today's ?) supply chain.

In essence, the simple RFID labels can not do much more than barcodes, and much more is actually not what we want from these. What is asked for is much **differently**. As seen in Subsection 2.2.1, the RF identification scheme allows objects to be tracked and recognized **on the move**, without even asking to be identified. It also allows the identification of objects even if the tags are not **in line of sight** of the readers. This is exactly the way to fit the needs of the modern assets tracking.

Even if this aspect is the main difference between these two technologies, other minor differences are also opening the path for new Auto-ID applications. The increased (and increasing...) storage capacity of the transponders and their readability under hard conditions are just two other advantages of the RFID systems.

Barcodes and RFID transponders are not alone in the Auto-ID market. Thus, an overall comparison of the systems is certainly a good way to identify the fields of use of each technology. For this purpose Table 2.5 provides a summary of the differences among these systems. Note that the facts summarized in this table are reflecting the state in 2005. As the Auto-ID field is really fast growing, some of the entries may quickly change. This is especially true for RFID systems and Smart Card technologies which are still the subject of many researches and studies.

<b>Parameter</b>	<b>Barcodes</b>	<b>OCR</b>	<b>Smart cards</b>	<b>RFID systems</b>
Data quantity [bytes]	1-100	1-100	16-64 Kbytes	16-64 Kbytes
Data density	Low	Low	Very high	Very high
Readability by humans	Limited	Simple	Impossible	Impossible
Influence of dirt or damp	Very high	Very high	Limited	No influence
Influence of the angle of reading	Limited	Limited	Limited	No influence
Degradation / wear	Limited	Limited	Contacts	No influence
Purchase cost of the medium	Very low	Very low	Medium	Low
Purchase cost of the reader	Medium-high	Medium	Medium	High
Operating costs	Low	Low	Medium	None to very low
Unauthorized copying / modification	Possible	Possible	Difficult	Difficult
Reading speed (whole medium)	Low (~4s)	Low (~3s)	Low (~4s)	Very fast (~0.5s)
Maximum distance between data carrier and reader	0-50 cm	< 1 cm	0 cm	30 cm-15 m
Typical distance between data carrier and reader	0 cm	0 cm	0 cm	1-2 m
Phase of use	Universally used (85% of the shops in the USA)	Widely used	Commonly used	In introduction, really fast growing

Table 2.5.: Comparison of some Auto-ID systems (from [Fin03] and [She04])

# 3

## Standards Around the EPC

---

<b>3.1. From the Auto-ID Center to the EPCglobal...</b>	<b>14</b>
<b>3.2. The Electronic Product Code</b>	<b>15</b>
3.2.1. Anatomy of an EPC tag	15
3.2.2. About Tag Data Translation	20
<b>3.3. The Physical Markup Language (PML Core)</b>	<b>21</b>
3.3.1. Elements of the PML Core	22
<b>3.4. The Savant</b>	<b>24</b>
3.4.1. The Event Manager	26
<b>3.5. The Object Name Service (ONS)</b>	<b>30</b>
3.5.1. ONS and DNS	31
3.5.2. EPC to DNS query	32
3.5.3. Results of an ONS query	32
<b>3.6. Towards an “Internet of Things”</b>	<b>33</b>

---

Having labeled or tagged objects being identifiable in an ubiquitous, seamless and flexible manner is already a good start. Building a network out of these objects, so that with a unique number we can easily retrieve information about the detected object, would enable much more interesting use cases. This is where the Auto-ID center, the EPC (Electronic Product Code) standards, ONS and PML all enter the scene, with a common goal of developing an Internet based system to identify assets anywhere in the world.

### 3.1. From the Auto-ID Center to the EPCglobal...

Created in 1999 as a federation of research universities (MIT, University of Cambridge, University of Adelaide, Keio University, Fudan University, and University of St.-Gallen), the Auto-ID Center had the challenging task of developing an open standard architecture for creating a seamless global network of physical objects (see [Aut]). After several years of research the **EPC Network** concept was born.

The Auto-ID Center ceased to exist in October 2003 with its mission done. The technology and works of the Center were transferred to an alliance between EAN international and the Uniform Code Council (UCC), called EPCglobal. Since October 2003, this non-profit organisation is responsible for the administration and the development of the EPC standards. The actors of the former Auto-ID Center are now regrouped under the name of Auto-ID Labs. Today, the Auto-ID Labs forms the research and development part of the field, whereas the EPCglobal is

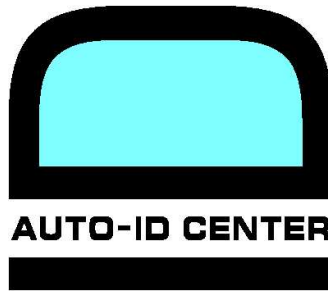


Figure 3.1.: Logo of the Auto-ID Center

in charge of bringing the technology and standards to the market.

Unlike the RFID technology, which is relatively old<sup>1</sup> (see Section 2.2), the EPC Network related researches are quite young. As a consequence, to date the only finalized standard of the EPC-global is the EPC Tag Data Standards Version 1.1 Rev.1.24 . The other specifications (Savant, ONS, PML) are still to be finalized, if ever adopted, as new standards are already on their ways. This is especially true for the Savant (see Section 3.4) which is in the process of being split into many independent standards.

## 3.2. The Electronic Product Code

The EPC uniquely identifies objects and facilitates tracking throughout the product's life cycle. This makes it similar to the Universal Product Code or an EAN Code (see Subsection 2.1.1), with the main exception that EPC was primarily designed to be efficiently referenced on networks. The EPC is the fundamental identifier of assets in a so called EPC Network. It basically contains information about:

- The manufacturer of the tagged object.
- The product class or the nature of the tagged object.
- The actual (unique) item. This latter information is an advantage over “classical” (i.e. UPC or an EAN-13) barcodes where two bottles of orange juice, from the same brand, of the exact same kind, have the same code<sup>2</sup>.

Each of the above information is encoded in a separate field which makes it quite easy to extract only part of the data. For instance, the sorting of items by manufacturers is made really straightforward by this latter fact.

### 3.2.1. Anatomy of an EPC tag

In the EPC-related literature, the word “EPC tag” is widely used. An EPC tag is basically an EPC compliant RFID transponder, that is a transponder which encloses in its chip a unique identifier of type EPC.

The data of an EPC tag is composed of three main parts as shown on Figure 3.2. A header, a filter (optional) that can be used to enable effective and efficient reading of the tags and a data field called the “Domain Identifier”. The EPC, or EPC Identifier is formed of the header and the domain identifier.

<sup>1</sup>Old in terms of “computer-years”...

<sup>2</sup>It is worth noting that second generation barcodes (i.e. UCC/EAN-128) already implement the possibility of having a serial number. For a summary of subject refer to [She04].

Standard EPC TAG DATA:

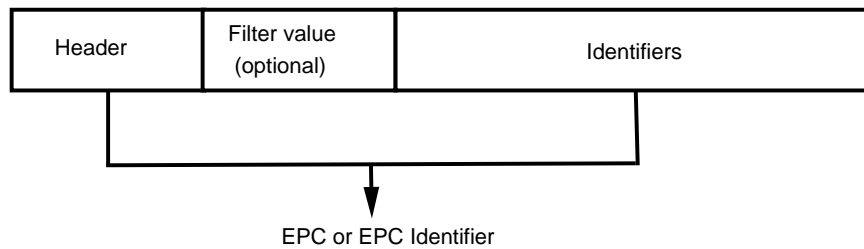


Figure 3.2.: The data of a standard EPC Tag

EPC identifiers can use various encoding schemes depending on the very nature of the industry it is enrolled in. According to the standard, one of the following coding schemes can be chosen [EPC04]:

- General Identifier (GID). It was created by the EPCglobal for applications that do not need to support any particular legacy encoding schemes.
- A serialized version of the EAN.UCC Global Trade Item Number (GTIN).
- EAN.UCC Serial Shipping Container Code (SSCC).
- EAN.UCC Global Location Number (GLN).
- EAN.UCC Global Returnable Asset Identifier (GRAI).
- EAN.UCC Global Individual Asset Identifier (GIAI).

An encoding scheme defines the syntax and semantics of the fields contained in the EPC. For instance coding an EPC using the Serial Shipping Container Code results in a EPC-SSCC 96 bits format. It contains two fields: the Company Prefix and the Serial Reference.

The coding scheme to be applied is defined in the header field of the EPC identifier. Enabling the use of various encoding schemes for EPC identifier (and not just the GID) ensures the compatibility of the system with legacy standards (mainly EAN or UCC barcodes). However, this fact complicates the whole process toward a global and interoperable EPC Network. To solve these issues, a Tag Data Translation working group has been created by the Auto-ID Labs. A brief overview of the challenges the group is facing and the solutions it already proposed is provided in Subsection 3.2.2.

The EPC identifiers and their coding schemes are standardized by a definition of the EPCglobal. At the time of writing this report the commonly used standard is the EPC Tag Data Standards Version 1.1 Rev.1.24(see [EPC04])

An EPC of type 1 (i.e. EPC Tag Data Standards Version 1.1 Rev.1.24 compliant) is encoded on 64 or 96 bits using one of the schemes described above. As the application developed for this work uses the GID encoding scheme, we will focus on this latter. An EPC-GID encoded on 96 bits has the following structure (see 3.3):

1. The header which is 8 bits long, defines the overall length and the format of the next fields. That is, it defines what exact encoding scheme was chosen. For instance the header 00110101 (or 35 in hexadecimal) means that the EPC is encoded using the GID-96 scheme.
2. The manufacturer field (28 bits) defines the actor in charge of the management of this particular EPC (e.g. a producer, a rental company, an organization etc.). It is worth noting that this part of the code is often known as the “EPC manager” field.



**Pure entities:**

This is the most commonly used URI explaining why it is sometimes called the “canonical form”. This URI contains the EPC fields required to distinguish an object from another. The pure entity representation for the EPC General Identifier is as follow:

```
urn:epc:id:gid:GeneralManagerNumber.ObjectClass.SerialNumber
```

where the three fields corresponds to the GID encoding scheme as described in Subsection 3.2.1.

In turn, a pure entity for the GRAI would look like this:

```
urn:epc:id:grai:CompanyPrefix.AssetType.SerialNumber
```

Thus, we can generalize the Pure Entities as follow:

```
urn:epc:id:ENC_SCHEME:SCH_FIELD1.SCH_FIELD2. ... .SCH_FIELDn
```

**URI for EPC Tags:**

For some applications it is useful to share information about the encoding schemes used on tags. Think of an application writing to RFID tags. Such a software need not only to be aware of what to write, it also needs to know **how** the writing should be achieved (i.e. using what scheme). One may notice that a Pure Entity URI already gives a brief information about the coding scheme. This is true but the information it gives is really basic. The maximal information that can be encoded in a Pure Identity URI is the encoding scheme and the number of bits contained in the code (e.g. `sgtin-96` or `sgtin-64`). Nothing is said about the required data fields. Coding this kind of information in a URI is the role of the EPC Tag URIs:

```
urn:epc:tag:EncName:EncodingsSpecificFields
```

`EncName` is one of the standards EPC encoding schemes (see Subsection 3.2.1). `EncodingsSpecificFields` includes:

- all the fields of the corresponding Pure Identity Type
- restrictions and numeric ranges (optional)
- the additional fields supported by the `EncName` encoding.

An example of a serialized GTIN 64 bits EPC Tag URI could be as follow:

```
urn:epc:tag:sgtin-64:3.0652642.800031.400
```

with the leading 3 being a filter value (see Figure 3.2).

**URI for invalid tags:**

Reading RFID transponders does not always lead to a success. Although the reliability of the tags is increasing really fast, errors will always remain. This is the purpose of the raw bit string URI. When no standardized EPC can be found on a tag, an “error log” may be sent to the application in the form of an EPC raw URI:

```
urn:epc:raw:BitLength.Value
```

Where the `BitLength` is the number of bits read and the `Value` is their decimal conversion.

Although one could be tempted to use this URI to communicate arbitrary (non-standard) bit strings, the EPCglobal recommends to use it **only** as a mean for reporting errors. Using it for anything else would result in a non-ability to distinguish errors from other streams. Besides, it would enable the companies to define their own standards which would significantly complicate and compromise the future of global EPC networks. Think of how complicated reading barcodes would appear today if each store were to have its own syntax and semantics...

**URI for EPC Patterns:**

The last type of URI is used for filtering purposes. It permits to identify sets of EPCs rather than single product codes. For instance, think of an application willing to get information only about two particular products, say: the BIO-apple-juice XYZ and the BIO-orange-juice ABC. The patterns it would use could look like that:

urn.epc.pat:gid:3.[10-11].\*

where 3 is the company number (assigned by EPCglobal as it has the authority on the GIDs), 10 and 11 correspond to our two products (BIO-apple-juice XYZ, BIO-orange-juice ABC), and the \* wild-card means any instance of these two products.

### URIs' Grammar Rules

To define the syntax of the URIs more precisely, the EPCglobal published a grammar rule for each of the resource identifiers. These rules, as well as a non-normative summary of the URIs' syntax, can be found in [EPC04].

As the EPCGID pure entity URI is going to be widely used in this paper it is worth briefly defining it. Let us start with a set of grammar elements which are common to every EPC-URIs representations:

---

```

NumericComponent ::= ZeroComponent | NonZeroComponent
ZeroComponent   ::= "0"
NonZeroComponent ::= NonZeroDigit Digit*
PaddedNumericComponent ::= Digit+
Digit            ::= "0" | NonZeroDigit
NonZeroDigit    ::= "1" | "2" | "3" | "4" |
                    "5" | "6" | "7" | "8" | "9"

```

---

followed by the EPCGID-URI syntactic definition:

---

```

EPCGID-URI ::= urn:epc:id:gid: 2*(NumericComponent . ) NumericComponent

```

---

which can in turn, be summarized using a non-normative representation:

urn:epc:id:gid:MMM.CCC.SSS

where MMM denotes the Manufacturer number, CCC the product class number and SSS the serial number of the labeled item.

### Summary of the Identity Concept

At this point, a brief summary of the identity concept will be of great help in understanding the combinations of the elements introduced in the current chapter.

To start with, the Identity Concept is the term naming the EPC Tag Data Standards. It is basically composed of three layers:

1. The Pure Identity layer.
2. The Encoding layer.
3. The Physical Realization layer.

To better understand the role of each of these layers we will introduce a simple example. Additionally, Figure 3.5 provides a brief overview of the layered architecture.

Given: 10656781007435, a GTIN (Global Trade Item Number) code with a serial number of 25. Its **Pure Identity Layer** can be represented as a **pure entity URI**. It will have the form:

urn:epc:id:sgtin:PPP.III.SSS

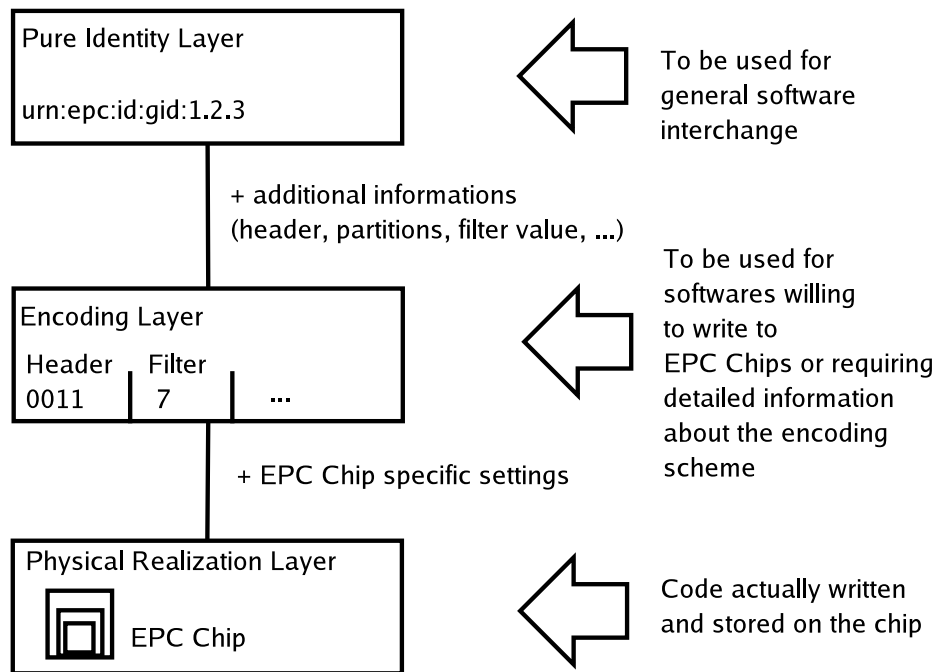


Figure 3.5.: Overview of the EPC's Identity Concept

where PPP denotes the EAN.UCC Company Prefix, III an SGTIN Item Reference and the SSS a Serial Number. In the URI, the indicator 1 is repositioned at the beginning of the item reference and the checksum digit 5 is dropped:

```
urn:epc.id:sgtin:0656781.100743.25
```

This Pure Identity URI is **independent of the physical type of the tag**. A fact that makes it the ideal candidate for software interchanges.

In turn, the **Encoding Layer** of our GTIN code can be represented as a GTIN-96 **EPC Tag URI**. Which would output as the following:

```
urn:epc.tag:sgtin-96:7.0656781.100743.25
```

As it informs about the exact encoding structure, this URI can be used in application dealing with chip writing or low-level tag operations.

Eventually, the GTIN identity has a **Physical Realization Layer**. This latter encoding is rendered in a machine-readable form suitable for a physical RF Tag. The 96 bits of the GTIN code written on a TAGSYS C240 RFID Chip is an example of Physical Realization.

### 3.2.2. About Tag Data Translation

As introduced in Subsection 3.2.1, the ability to translate one coding scheme to another is a really important issue. The EPC Network aims to be global. Thus, one choosing a coding scheme should be able to translate it into whatever code accepted by the standard. However, this translation is far from being trivial. The differences between the semantics of the codes is one out of many issues that the translator should solve.

This is why the Tag Data Translation Working Group was created. These members of the Auto-ID Labs are working on an upgradable conversion engine usable at any Layer of the EPC (or EPCglobal) network [Har]. A prototype of the engine, written in Perl and Java, is available from [36].

### 3.3. The Physical Markup Language (PML Core)

Besides the EPC, which is the fundamental component of an EPC Network, the Physical Markup-Language (PML) is also of great importance. As defined in [FAOH03] the “PML is a collection of vocabularies used to represent information related to EPC network enabled objects”. The most important of its vocabularies is certainly the PML Core, which is described in the following section.

The version of the PML specification on which this paper is based is known as PML Core Specification 1.0. It was published by the former Auto-ID Center in 2003. The objective of the PML Core is to provide a standardized format for information interchange within an EPC Network. It is used to model and encapsulate the data captured by the Auto-ID sensors (i.e. RFID readers or antennae), providing a generic markup-language. The PML Core language is based on an XML syntax validated by two XML-Schema:<sup>4</sup>

- `PmlCore.xsd`, which contains the Sensor related elements.
- `Identifier.xsd`, which provides a validation schema for unique identifiers. It is worth noting that the use of the EPC scheme as the identifier within PML files is strongly encouraged but not mandatory.

Appendix B.1 provides a copy of both `PmlCore.xsd` and `Identifier.xsd`.

Before going on deeper into the subject, let us consider a simple example, emphasizing the concrete use of the PML. It will also demonstrate using a really basic example, a use case of the Auto-ID technologies within the supply chain.

Given two independent companies, *A* and *B*, equipped with RFID EPC assets trackers. *B* is a transport company. *A* is manufacturing and selling orange juice all over the world.

To simplify its supply chain management and reduce the costs, *A* enrolls *B* for driving the goods to foreign countries. *A*, still wants to be informed of where the goods are. This means *B* will have to inform *A* about where the goods are, on a regular basis. A straightforward way of achieving this would be for *B* to send short messages to *A* each time a box of orange juice is detected by one of its RFID readers. Such a message might look as shown on Figure 3.6.

A quick read of this message clearly emphasize the need for this information to be standardized. Giving a formal way of writing such messages is the exact goal the PML Core is attempting to achieve. It provides a very generic semantic and syntax to encapsulate the sensors’ observations. Object observations (such as RFID tags), as well as physical observations (temperature, GPS location, etc.).

The 80/20 rule<sup>5</sup> as published in [FAOH03] expresses a fundamental element of the PML’s basic philosophy:

“The design of the PML Core should provide 20% of the features that accomodate 80% of the needs.”

It permits the PML to stay generic and simple, not related to some companies’ special needs.

One may also notice that the PML was thought independently of any recording support. This basically means that this markup language is not necessarily the one to be recorded in the companies ERPs. The institutions are free to persist the sensor’s observation in any manner they can think of. The PML can then be generated on the fly, whenever formalized information interchange is required.

<sup>4</sup>The XML schema version 1.0 were used for the PML Core Specification 1.0.

<sup>5</sup>The 80/20 rule is based on Pareto’s principle, stating that 20% of the Italian population owned 80% of the goods.

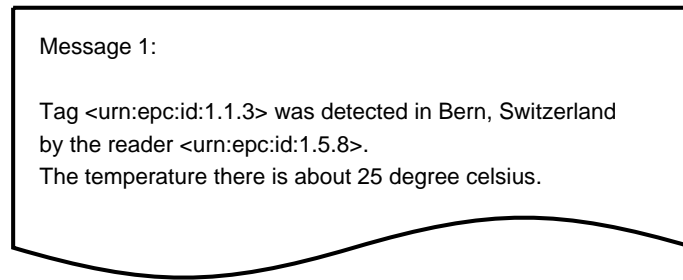


Figure 3.6.: A simple, non-formalized, goods tracking message

### 3.3.1. Elements of the PML Core

The PML Core defines a set of elements used to report the sensors' observations called the PML Core Sensor Data Model. The allowed combinations within a PML document are given by Figure 3.7.

The following list gives a summary of the use of each element:

- The Sensor element is the root of a PML document. It represents an Auto-ID sensor. A typical example of sensor would be an RFID reader or, to be more precise, the antenna of an RFID reader. The Sensor root is composed of an ID element as well as one or more Observations.
- The ID element serves as an identifier for the physical elements (tags, sensors, antennae etc.) of the EPC Network. It is standardized by the Identifier.xsd XML schema. By default, the ID element contains an identifier of type EPC in the form of an EPC URI (see Subsection 3.2.1).
- The Observation element contains the result of a sensor's measurement. It comprises at least a DateTime element but is usually enclosing at least one Tag or Data element as well. An Observation can contain an ID. In this particular case, the ID is not meant to be an EPC. It is more likely to be an integer intended to number the Observations.
- The Command element can also be part of an Observation. This latter contains the type of the event (e.g. TagsIn, TagsOut, etc.) that occurred. The set of possible values for this element is sensor-dependent.
- The DateTime element is an XML schema dateTime element written in the form CCYY-MM-DDThh:mm:ss. It can contain some additional options (e.g. GMT, decimal fraction of a second, etc. See [HM02] for a complete list). For example, 13:50 (GMT+3), December 31, 2008 should be written as 2008-12-31T 13:50:52.292+03:00.
- The Tag element is the actor of an Observation. It represents an identification label detected by the ancestor Sensor. In the case of the concrete application (the RFIDLocator) described in this paper, the Tag element is going to be an RFID-EPC label manufactured by Philips Rafsec. A Tag element consists of the followings:
  - An ID. In our case this latter is going to be of type Electronic Product Code.
  - An optional Data element.
  - Zero or more Sensor elements. These are intended for **sensors mounted on tags** such as temperature or altitude indicators. It is worth noting the recursive structure of such an element enclosed in a Tag.

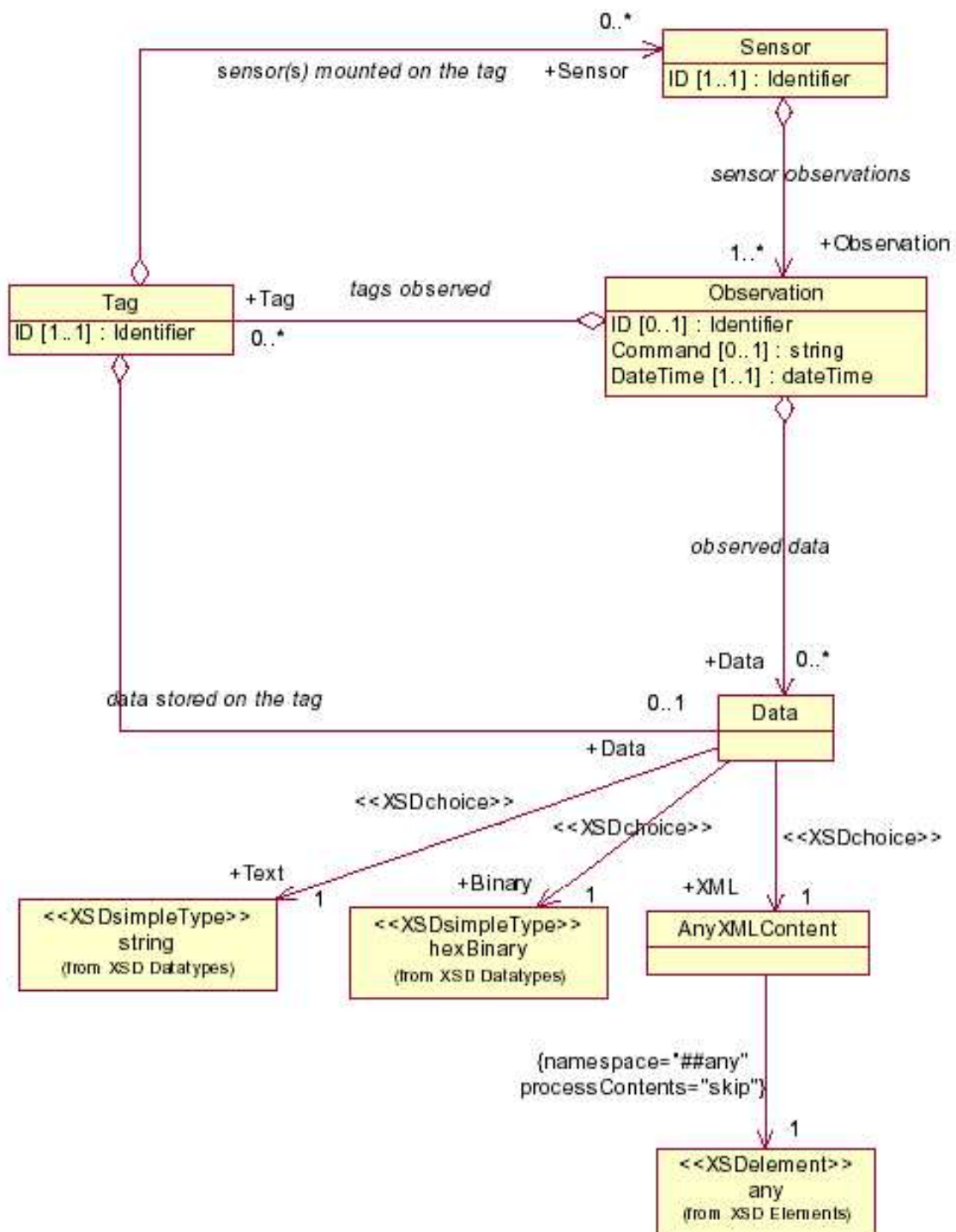


Figure 3.7.: The sensor data model (from [FAOH03])

- Finally: the **Data**. This last element is used to represent non-tag observations (temperature, speed, altitude, location, etc.). It consists of **one** of the above elements:
  - A Text, which is nothing but an XSD “string”.<sup>6</sup>
  - A Binary, element which is a hexadecimal number and uses the “hexBinary” XML schema data type.
  - An XML element. This latter can contain XML tags which can be useful to extend the formalism of the observations.
  - Eventually, it can contain an XSD “any” element. This last possibility, as well as the former, provides developer with a way to extend the Data element and thus, the PML.

To end this short overview of the PML Core, a concrete example is given in Figure 3.8. It was extracted from the architecture built for the RFIDLocator. The sensor is a PML-Reader, which is an RFID reader simulator designed by SUN Microsystems and bundled with the Sun Java System RFID Software v.1.0. The extract is extensively commented for a better understanding of the PML Core basic elements. It basically reports an Observation of 3 tags:

- urn:epc:id:gid:1.1.101
- urn:epc:id:gid:1.1.102
- urn:epc:id:gid:1.1.103

entering the field (TagIn) of the Physical Antenna (or Sensor) identified by urn:epc:id:gid:1.255.1.

### 3.4. The Savant

As a preliminary remark it should be acknowledged that, at the time of writing, the term Savant<sup>7</sup> had been deprecated [GS04] and its related specification were in the process of being split into many elements (Event Manager, EPCIS, Web Services etc.). However, since the Sun Java System RFID Software v.1.0 is Savant 1.0 compliant, it still does make sense to give a summary of the Savant’s specification. Such an overview also has a value to understand the changes to come in the EPC Network architecture.

The following sentence found in [Goy03] is a good way of introducing the Savant: “If each object currently using a barcode were to make the use of [...] EPC, then the managing infrastructure would have to handle millions of events every second [...]”

Thus, the Savant is a set of modules solving this important issue. According to [Goy03] as well as to the standard, the Savant has three main components:

- The Real-time In-memory Event Database (RIED) an optimized database. This component offers much better performances than a standard database when more than a few hundred transactions per second occur.
- The Task Management System (TMS), is a generalized scheduling manager. It can be used to command the execution of various tasks in the Savant. It also contains modules for restarting tasks after a failure.
- The EMS (or Event Management System), in charge of managing the readers’ output flows and preparing it for higher-level applications.

<sup>6</sup>I.e. an XML schema “string” data type.

<sup>7</sup>The Savant is a registered trademark.

---

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- In this example, the Sensor was the PMLReader from Sun Microsystems -->
<Sensor xmlns="urn:autoid:specification:interchange:PMLCore:xml:schema:1">
  <ns1:ID
    xmlns:ns1="urn:autoid:specification:universal:Identifier:xml:schema:1">
    <!-- the PMLReader is identified using its own EPC -->
    urn:epc:id:gid:1.255.1
  </ns1:ID>
  <!-- Observation = event occurred within the scope of the readers antennae -->
  <Observation>
    <ns2:ID
      xmlns:ns2="urn:autoid:specification:universal:Identifier:xml:schema:1">
      <!-- this observation was the second to occur, this ID is not an EPC -->
      2
    </ns2:ID>
    <!-- Standardized event occurrence time -->
    <DateTime>2004-12-28T13:46:52.292+01:00</DateTime>
    <!-- Command = type of the event -->
    <Command>TagsIn</Command>
    <!-- 3 RFID tags initialized the event, each one of them is referenced
    in a "Tag" container -->
    <Tag>
      <ns3:ID
        xmlns:ns3="urn:autoid:specification:universal:Identifier:xml:schema:1">
        <!-- The identifier of the tag is expressed using its pure
        entity URI -->
        urn:epc:id:gid:1.1.101
      </ns3:ID>
    </Tag>
    <Tag>
      <ns4:ID
        xmlns:ns4="urn:autoid:specification:universal:Identifier:xml:schema:1">
        urn:epc:id:gid:1.1.102
      </ns4:ID>
    </Tag>
    <Tag>
      <ns5:ID
        xmlns:ns5="urn:autoid:specification:universal:Identifier:xml:schema:1">
        urn:epc:id:gid:1.1.103
      </ns5:ID>
    </Tag>
  </Observation>
</Sensor>

```

---

Figure 3.8.: PML extract generated by a PML-Reader

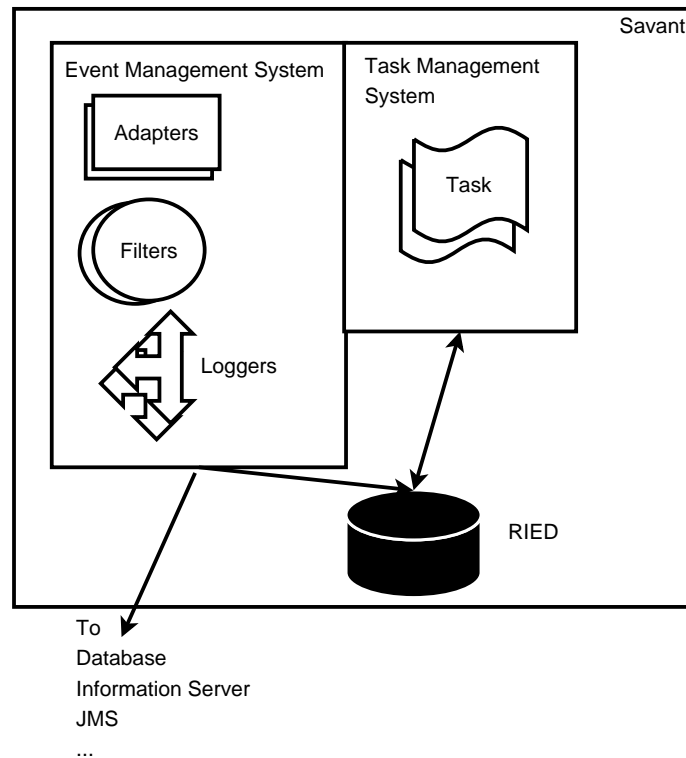


Figure 3.9.: Components of the Savant

Figure 3.9 offers a view of the component’s organisation within the Savant. When talking of the Savant, the concrete work in this project was done on the EMS (aka Event Manager or EM). This is why the following subsection focuses on this latter part.

### 3.4.1. The Event Manager

The Event Manager is somehow like a tub of Lego<sup>8</sup> bricks. It is composed of many modules that are combinable to form an aggregate. The structure of the EM fits the Hollywood Principle<sup>9</sup>. This fact, among other properties, makes it a framework providing methods and tools to manage and capture EPC events. The main elements of the EM are the followings:

- **The Adapters:** these are the device drivers. They are vendor-specific hardware drivers in charge of capturing the events out of these machines. As an example the Sun Java System RFID Software comes bundled with Adapters for the Alien, ThinkMagic and Matrics RFID readers (see [Sun04a] for further details). However, using the Sun Java System RFID Software with other devices is possible. For instance the RFIDLocator uses a Tagsys Medio L100 for which an adapter was programmed by Tagsys.
- **The Filters:** these modules are responsible for the processing or the skipping of the received events. Just about any type of filter can be designed. Figure 3.11 gives a code prototype of a custom-made filter. Technically speaking, it is worth noting that a filter in the Savant has to implement the `EventFilterInterface` as defined in the Savant Technical Manual (see [4]). Additionally, Table 3.1 gives an overview of the filters shipped with the Sun Java System RFID Software. One could notice that the `EPCFilter` in this Table is a concrete implementation of the filter described in Figures 3.11 and 3.12.

<sup>8</sup>Lego is a registered trademark of Lego Inc.

<sup>9</sup>The Hollywood principle can be summarized by the sentence: “Do not call us, we will call you !”

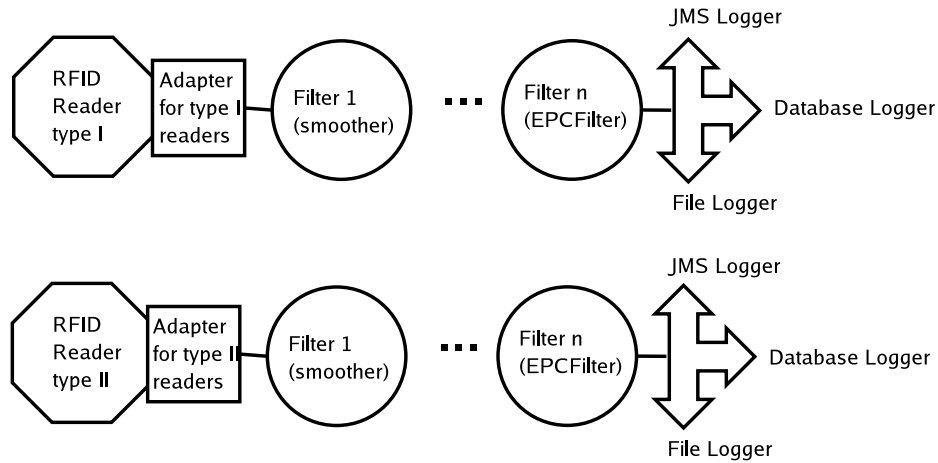


Figure 3.10.: A example of modules combining in the EM

Name	Description
BandPass	Event matching a particular event of the reader (i.e. TagsIn) are forwarded.
Delta	Tags are reported only on entering and leaving the field of view.
EPCFilter	Filters the tag on a part of their EPC (e.g. forwards only the events about a particular product).
Smoothing	Empowers the reader's accuracy by comparing successive read cycles.

Table 3.1.: Filters provided by Sun in the RFID software solution

---

```
import org.autoidcenter.ems.*;
import org.autoidcenter.exception.EPMFException;

/* This filter forwards all the non-epc and status events it
 * receives. When receiving epc-events, it forwards only those
 * events that start with 00000000B0000030000. This EPC beginning
 * may represent a particular product or range of products.
 *
 * Source: http://www.autoidlabs.org, Amit Goyal
 * Commented by: Dominique Guinard */

public class RangeFilter implements EventFilterInterface {
    private ReaderInterface loggers[ ];

    /* The constructor of this filter sets the loggers variable
     * to null. This variable will be set by an external call if a
     * logger has to be used at the output of this filter. */
    public RangeFilter( String args ) {
        loggers = null;
    }

    /* what should happen when an EPC event is raised */
    public void logEpcEvent( long timestamp, String EPC,
        String readerEPC ) {
        /* check if the EPC matches our requirement */
        if ( EPC.startsWith( "00000000B0000030000" ) )
        {
            if ( loggers != null )
            {
                /* if it matches and at least a logger is
                 * registered, then forward the event to all
                 * the loggers.
                 */
                for ( int i = 0; i < loggers.length; i++ )
                {
                    ReaderInterface logger = loggers[i];
                    /* The time of the event as well as
                     * the EPC of the reader detecting it are
                     * sent to the loggers
                     */
                    logger.logEpcEvent( timestamp, EPC,
                        readerEPC );
                }
            }
        }
    }
}

/* [...] */
```

---

Figure 3.11.: Implementation of a simple filter using Java (first part) (from: [Goy03])

---

```
/* continuing ... */

/* Events other than EPC related can occur (e.g. the
 * read on a non EPC compliant Tag), here is what to do
 * with such an observation. */
public void logNonEpcEvent( long timestamp, String readingType,
String value, String readerEPC ) {
    if ( loggers != null ) {
        for ( int i = 0; i < loggers.length; i++ ) {
            ReaderInterface logger = loggers[i];
            logger.logNonEpcEvent( timestamp,
readingType, value, readerEPC );
        }
    }
}

public void logStatusEvent( long timestamp, String statusMessage ) {
    if ( loggers != null ) {
        for ( int i = 0; i < loggers.length; i++ ) {
            ReaderInterface logger = loggers[i];
            logger.logStatusEvent( timestamp,
statusMessage );
        }
    }
}

/* function used to set loggers at the output of the current
 * filter. */
public void setListeners( ReaderInterface loggers[] )
throws EPMFException {
    this.loggers = loggers; }
public void shutdown() {}
}
```

---

Figure 3.12.: Implementation of a simple filter using Java (second part) (from: [Goy03])

Name	Description
FileLogger	Forwards the events (in a PML Core stream) to a specified location on the file-system.
HttpPMLLogger	Sends the events (PML Core) to an HTTP connection.
JMSLogger	Forwards the events in asynchronous PML messages using the JMS API. This logger was chosen for the RFIDLocator presented in this report.
SocketLogger	Creates a socket connection and forwards the events (PML Core) to the specified port.
SsocketLogger	Similar to the SocketLogger but creates a server socket connection.

Table 3.2.: Loggers available in the Sun Java System RFID Software

- **The Loggers:** are used to actually monitor, forward or store the events. They are the end points of the Savant's chained structure. The Sun Java System RFID Software v.1.0 provides a variety of loggers as described in Table 3.2.

On top of these elements, business applications can be quite easily built. Such software need no more to be aware of the Tag capture technical details. It just needs to implement a way to listen to a logger. As a summary of this section, Figure 3.10 gives a typical way of combining the modules of the Event Manager.

### 3.5. The Object Name Service (ONS)

At this point of the EPC Network exploration we have:

- A standardized way to uniquely identify labeled objects: the EPC;
- A standardized language to interchange sensors' observations within the EPC Network: the PML;
- A framework to handle the EPC events: the Savant.

This means we are close to the goal of creating an EPC Network. However, at least one important element is still missing. The EPCglobal wants the EPC Network to be global. As a consequence, when an EPC is detected we should be able to find authoritative information about it. This should be the case whoever owns this information and, maybe even more important, **wherever** this piece of information is stored.

For instance, consider the example of Section 3.3. To recall the situation: a corporate *B* was in charge of the transportation of orange juice for the company *A*. Now imagine a truck of *B*, full of bottles from *A*, arriving at the customs. The officer on duty needs to know the origin of the oranges that were used to produce the juice. As the freight was not produced by the *B* company, the driver does not have this information (assume that the communication level between *A* and *B* is not yet at its paroxysm...). This is where the ONS enters the game. Using an EPC reader connected to a computer, the customs officer scans the truck getting the EPC

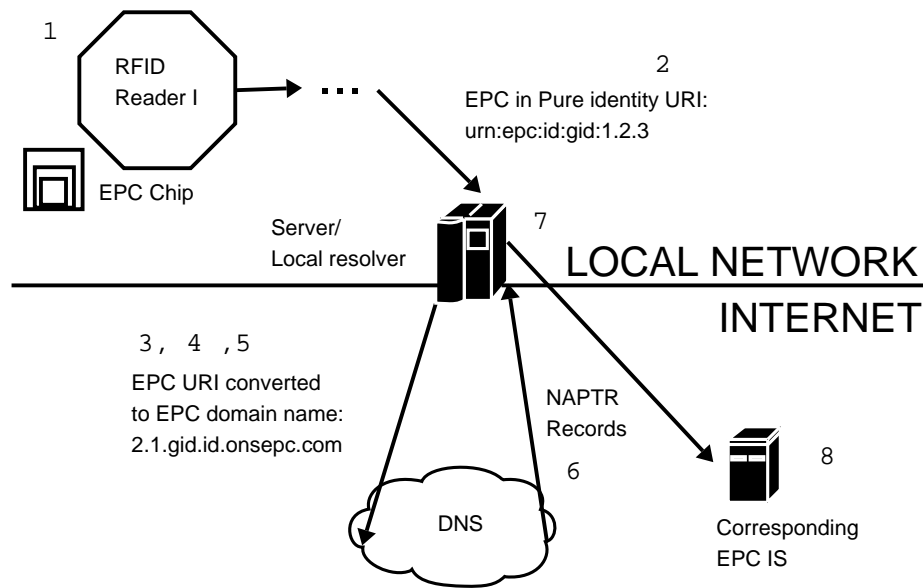


Figure 3.13.: An ONS query by a client application

of the bottles. These EPC are then sent by the reader to an Object Name Service (through the Internet) which returns the corresponding URL: <http://www.theAcompany.com/cgi-bin/products.cgi?product=OrangeJuice&type=SunnyJuice>. It is retrieved by a client application which opens a browser and displays information about this particular product.

The driver hasn't lost his day, and the customs officer has got accurate information to fill his forms...

This fairly simple example leads to a first definition of the ONS: **Given an EPC, the ONS provides URLs to authoritative information** [Aut04]. The ONS and its development are managed by the EPCglobal. However, to date, there is no finalized standard specification. This explains why this section is based on a (advanced) Working Draft of the EPCglobal (see [Mea04]).

### 3.5.1. ONS and DNS

As seen in the example above, ONS works similarly to the well known Domain Name Service (DNS). This latter was initially intended to map host names (e.g. <http://www.gnipsoft.com>) to their IP address (e.g. 213.173.189.45). It has since been extended to a wider variety of Internet services (like matching URLs to phone numbers, etc.).

The similarities between DNS and ONS are not void. In fact ONS uses **the Domain Name Service** to retrieve the required information. Figure 3.13 describes an ONS query in the view-point of a client application (see [Mea04]). Such a query goes through the following steps:

1. An EPC tag is read by an RFID Reader
2. After a pass in the modules of the Event Manager (see Subsection 3.4.1) the URI encoded EPC is made available.
3. A client application (on the server) starts an ONS query to retrieve information about the labeled product.
4. The resolver (on the server) translates the EPC pure entity URI into a DNS readable domain name. This step is detailed in Subsection 3.5.2.

5. The resolver issues a DNS query for NAPTR records. These Naming Authority PointeRs are record sets containing the information a DNS sends back after a successful query. Their structure is discussed below.
6. The DNS infrastructure returns the NAPTR records containing URLs pointing to one or more Internet services.
7. The local resolver extracts the URLs from the records and sends them back to the local server on which the client application resides.
8. The application scans the list for the service (e.g. EPC IS, HTML server, etc.) best fitting its needs and contacts it.

The most important of the above elements are going to be further described in the following subsections.

### 3.5.2. EPC to DNS query

A Domain Name Service does not understand EPC URIs. As a consequence, in step 4 of Figure 3.13, the EPC pure entity URI needs to be translated into a domain name form. This is achieved by going through the following steps (see [Mea04]). To illustrate the process consider the following EPC URI: `urn:epc:id:gid:1.2.3`.

1. Remove the “urn:epc” header, giving `id:gid:1.2.3`.
2. The version of ONS on which this section is based stops the offered queries at the object class (see Subsection 3.2.1).  
As a consequence, ONS queries are not specific to an instance of a particular product but to the class of a product: e.g. specific to the “Fresh orange juice Premium” from the company A but not to a particular bottle of it.  
This basically means that the serial number has to be removed of the URI, giving `id:gid:1.2`.
3. The order of the remaining fields is inverted, we now have `2.1:gid:id`.
4. Convert all the “:” to “.”, leaving: `2.1.gid.id`.
5. “onsepc.com” is eventually appended to the string giving: `2.1.gid.id.onsepc.com`. Using this latter string, the client’s DNS solver is used to query for NAPTR records.

### 3.5.3. Results of an ONS query

As explained in the previous sections, the results of a ONS query are sent in the form of NAPTR records. Various (somehow simplified) examples of such records are presented by Table 3.3.

The `order` field is used to ensure the records are interpreted in the correct order. In a ONS query, since all the records are valid for the EPC in question, it helps finding equivalent records. For instance, records 1 and 2 of Table 3.3 both have an `order` of 99 and offer the same service thus, they are equivalent and can be used interchangeably.

When entries are equivalent, the `pref` field gives the preferred record. As an example consider, once again, the records 1 and 2 of Table 3.3. If the first service (`pref: 0`) was to be out of order, the client application would try using the service listed on the second line (with a `pref` of 1). In short, the first two fields of an NAPTR record are used for load-balancing purposes.

The `service` field is more interesting. It describes the class of the proposed service (i.e. the nature of the server). This field is standardized and its values can not be arbitrarily chosen according to a company’s needs. A set of possible values was issued by EPCglobal :

Rec. id	Order	Pref	Flags	Service	Regular Expression
0	99	0	u	EPC + html	!^.*\\$\$!https://www.theAcompany.com/products/ordering/order_bio.php!
1	99	1	u	EPC + html	!^.*\\$\$!https://mirror.theAcompany.com/products/ordering/order_bio.php!
2	98	1	u	EPC + epcis	!^.*\\$\$!http://www.theAcompany.com/epc/cgi-bin/epcis.cgi!
3	97	2	u	EPC + ws	!^.*\\$\$!http://www.theAcompany.com/epc/ws/bio_orange.wsdl!
4	96	0	u	EPC + xmlrpc	!^.*\\$\$!http://rpcprovider.xmlrpc.com/clients/theAcompany.com!

Table 3.3.: A typical set of NAPTR records

- **EPC+ws**: this first class of services opens the door to Web Services. With all the power and interoperability that they enable.
- **EPC+epcis**: represents an EPC Information Service. Such a server returns authoritative meta-data about a specific product. For instance, an EPCIS could return raws of tag’s observations in a PML Core format (see Section 3.3).
- **EPC+html**: is used to represent an URL to an HTML (HTML, ASP, PHP, etc.) page on some server. The page should contain information about the EPC in question. Assuming that a company already has a website then using ONS to offer EPC HTML services is really straightforward. There is not need for the company to modify their systems. The only required step is to match ONS requests to already existing web-pages.
- **EPC+xmlrpc**: is used to denote a service expecting XML-RPC compliant connections.

This short list reflects the initial services registered by the EPCglobal to be used with the ONS, as specified in [Mea04].

The fact that the service field is managed by the EPCglobal represents an obstacle to a fast innovation. However, it avoids the ONS system to be fed with proprietary, closed or non-standards services. This latter fact ensures the interoperability of services returned after an ONS query. It is to note that a special class of services was meant for **experimental applications**. Indeed, a service starting with x- is allowed not to be registered.

This section was meant as an introduction to the ONS facilities, a more complete overview of the subject can be found in [Mea04].

### 3.6. Towards an “Internet of Things”

From the very beginning the vision of the Auto-ID Centers was quite clear: create and develop what they call a “Smart World”. Understand: an infrastructure based on computer networks linking together (physical) objects, information and people. The primary goal of this infrastructure is to enable universal coordination of physical resources through remote monitoring and control by humans or machines [Goy03].

According to the Auto-ID Center to fully achieve this goal, the future EPC network will need to meet four main constraints:

1. The system should be able to network objects without human involvement.

2. The networking of objects should be seamless, i.e. operate continuously and not just at check out or at check in.
3. The network should be inexpensive.
4. The network would have to be ubiquitous, enabling objects to be detected in various environments and not just at manufacturing plants or distribution centers.

Looking closer at these constraints we can see that see the RFID technology meets (or will meet) most of them. Thus the EPC standards coupled with the Radio Frequency IDentification might well make the dream of an “Internet of Things” a reality.

Using standardized identifiers (EPC), exchange formats (PML), query systems (ONS) and events processors (Savant) we are virtually able to connect every single object of this world to a global network. Indeed, if you paste an RFID label onto the table of your living-room and match the description of your table with the enclosed EPC, your table becomes part of a global network. Starting from this point, your table can be identified and traced by computer systems coupled with RFID readers.

This simple idea of matching a computer-readable standardized number to any object surrounding us has an incredible number of applications.

As seen in this chapter the EPC Network is still young but its potential might well boost the number of researchers and manufacturers around the technology. Leading to a global EPC Network earlier than one may think.

**Part II.**

# **The RFIDLocator Application**

# 4

## Introduction to the Application

---

<b>4.1. Introduction to the Idea</b> . . . . .	<b>36</b>
<b>4.2. Introductory Terminology</b> . . . . .	<b>37</b>
<b>4.3. Use Cases</b> . . . . .	<b>39</b>
4.3.1. Main Use cases . . . . .	39
4.3.2. Demonstration Use Cases . . . . .	44

---

### 4.1. Introduction to the Idea

As exposed in this paper’s introduction, the main goal of this Bachelor Thesis is to develop an application demonstrating both the uses of the RFID/EPC technologies and a use case of the Sun Java System RFID Software. At this early point in the exploration of the software, it is worth noting that the RFIDLocator is not a fully featured application but a working prototype. The basis of the idea behind the RFIDLocator is quite simple but still satisfies, we believe, a concrete need in many application domains: “Given a set of objects moving (intensively) in a predefined area, where can we find them at a given time  $t$ ”.

The reader should understand at this point that the RFIDLocator is not intended for assets tracking within the supply chain (i.e. RFID’s favorite field) but for locating objects within a given area.

To make this important difference more obvious let us provide some use case examples of this software.

#### The law firm

Imagine a law firm: Law and Co. with 20’000 clients for 100 lawyers. As in the law business paper is still quite relevant, the company stores two physical files (filled with contracts, important documents, etc.) for each of its clients’ cases. Many lawyers are working on one case. As a consequence the files travel from one office to the other many times a week. Thus, the lawyers spend quite a part of their days looking for the documents within the 20 floors the company occupies.

A straightforward solution to Law and Co’s problem would be to hire less qualified (and thus, less paid...) employees to do the seeking job for the lawyers. Another solution could be to buy a dozen of RFID Readers, paste RFID labels onto each physical file and setup a software like the RFIDLocator.

The lawyers would then only need to query the system in order to get the location of Mister Steel’s case.

This use case might seem somehow artificial. However this is not the case as it reflects a real need for an automatized way of tracking documents within the buildings of a company. The dubious reader is invited to look at a concrete example summarized in [Sco].

### The Hospital

The tracking of documents within the buildings of an institution also makes sense when talking about a hospital, where finding a medical file is sometimes a matter of life and death. The real need for such an application in this field was confirmed by a small interview of a medical secretary<sup>1</sup> realized by the author of this Bachelor Thesis.

### The Museum

Consider the tracking of fossils in a well sized archaeological museum. Using RFID tags, readers and a tracking software like the RFIDLocator one could easily find out by who (and where) the giant-turtle fossil is being studied at the moment.

### The Data Center

For this last example we consider the case of a data center. Placing RFID labels on each server as well as readers at the entrance of each room, would permit to track the servers. Indeed, using an application like RFIDLocator one would then be able to find in what room (or even rack) a particular machine is located.

This rest of this chapter relates the background and the idea of the application by exposing its use cases.

To start with, some of the terms widely used in the application and its APIs are defined.

## 4.2. Introductory Terminology

Word	Meaning
Action	<p>In order to trace Business Object, two actions are provided by the application:</p> <ul style="list-style-type: none"> <li>• IN: Used when a <code>TraceableObject</code> is entering a given <code>Location</code>.</li> <li>• OUT: To be used when a <code>TraceableObject</code> is leaving a given <code>Location</code>.</li> </ul>

<sup>1</sup>Miss Evelyne Beaud from the Hôpital Universitaire de Genève (see [13]), Switzerland was interviewed.

Word	Meaning
Antenna characteristics	The characteristics of an antenna are not primarily meant in terms of the physical capabilities of the enclosing device (e.g. tags reads/sec, power supply, Operating System, etc.), but they include information about the surroundings of the antenna. In short, the characteristics contain all the information that is useful to the RFIDLocator (e.g. the Location of the antenna, the reliability of the device, etc.).
Business Number	A Business Number is given by a Business Object numbering policy within a company. The RFIDLocator is not in charge of checking the uniqueness and the semantic of a Business Number, it just uses it as provided.
Business Object	A Business Object <sup>2</sup> is the business-related representation of a TraceableObject. Managing and creating such objects is not in the scope of the RFIDLocator. The most important point about a Business Object resides in the fact that it <b>must</b> be assigned a unique Business Number.
Business Location Number	A Business Location Number refers to the corporate “geographical” numbering policy. It provides a unique (firm’s wide) way of identifying locations. Examples of such numbering schemes are room numbering (e.g. RM-3.113 which refers to a third floor office in the building ”Regina Mundi” at the University of Fribourg), or owner/rooms pairs (e.g. “the office of Miss Gachet”).
Location	A Location is a <b>place</b> (e.g a building, a room, a rack, etc.) within the area where the RFIDLocator is deployed. It is identified by a unique Business Location Number.
LogicalAntenna	A LogicalAntenna is an aggregate of 1.. <i>n</i> PhysicalAntennae. It is used to determine what Action (i.e. IN/OUT) was effectively recorded by the <i>n</i> PhysicalAntennae. A LogicalAntenna is always placed in a Location.

<sup>2</sup>Physical Object, Object, File are all synonyms of a Business Object.

Word	Meaning
PhysicalAntenna	A PhysicalAntenna represents the hardware able to capture EPC events by emitting a radio frequency field. As such an antenna is not standalone, it is always connected to a Reader. A PhysicalAntenna must be identified by an EPC (or another type of unique identifier).
Reader	A Reader is a physical hardware device controlling a set of 1..n PhysicalAntennae grouped in LogicalAntennae.
Sensor	A sensor is a synonym (introduced by the EPC-global) for a PhysicalAntenna.
TraceableObject	A TraceableObject is the fundamental element of the RFIDLocator. It represents a physical object that was tagged with an EPC label. Examples of TraceableObjects could be a medical file in a hospital or a case in a law firm. It is worth noting that the demonstration application will focus on files (aggregates of physical documents) which are just one type of TraceableObject.

## 4.3. Use Cases

### 4.3.1. Main Use cases

This section exposes the use cases of the core application. Using them, one is able to register a physical object and track it within a given set of buildings. The use cases that were actually implemented in the RFIDLocator are signaled with the keyword **implemented**. Additionally the Figures 4.1 and 4.2 expose the use cases using the UML formalism.

#### Attach Tag (implemented)

- **Actor(s):** User
- **Goal:** Matches an EPC or a unique identifier (the one recorded on the tag pasted onto the Business Object) to the Business Number of a TraceableObject.
- **Precondition:** The User (employee) is registered and logged in.  
The TraceableObject is tagged with a valid EPC label, the following information are known:
  - Business Number of the Business Object.
  - Current Location of the Business Object.
  - Value of the EPC pasted onto Business Object in a URI form (e.g. urn:epc:id:gid-96:12.10.3).
- **Workflow:** Using the “Attach/Detach” page, the user enters both the EPC URI and the Business Number that are matching together.

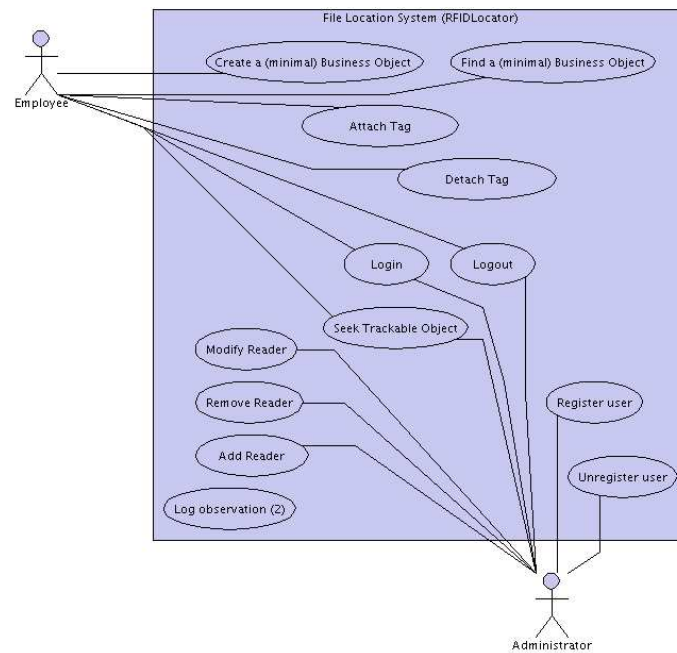


Figure 4.1.: Use cases on the Application Server (core application)

- **Postcondition:** The Business Object (thus, becoming a TraceableObject) is virtually attached to its EPC or an exception is thrown and reported to the client.

#### Detach Tag (implemented)

- **Actor(s):** User
- **Goal:** Removing a TraceableObject that is no more available. Detaching a Business Object from its EPC makes it possible to reuse the EPC label for another Business Object <sup>3</sup>.
- **Precondition:** The User (employee) is registered and logged in.  
The TraceableObject was previously registered in the system.  
The Business Number of the TraceableObject to detach is known.
- **Workflow:** Using the “Attach/Detach” page, the user selects “Detach” and enters the Business Number of the TraceableObject to detach.
- **Postcondition:** The TraceableObject and its Observation are removed or an exception is thrown and reported to the client.

#### Seek TraceableObject (implemented)

- **Actor(s):** User
- **Goal:** Locates a registered TraceableObject.
- **Precondition:** The Business Number of the TraceableObject is known and the TraceableObject was previously registered (see Attach Tag).  
The User (employee) is registered and logged in.

<sup>3</sup>It is worth noting that this removes the TraceableObject of the RFIDLocator only and not of the existing business systems.

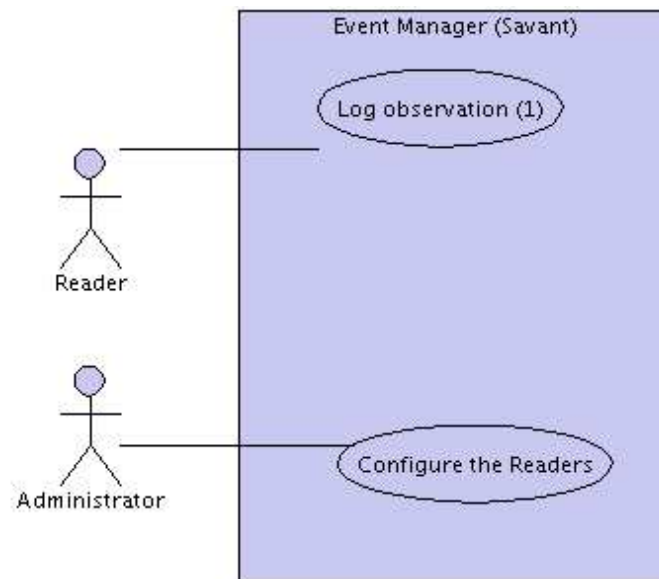


Figure 4.2.: Use cases on the Event Manager

- **Workflow:** Using the “Seek traceable object” page the user enters the Business Number of the TraceableObject to look for.
- **Postcondition:** The latests observed locations of the TraceableObject are returned (as well as additional information such as: the action, a timestamp...) or an exception is thrown and reported to the client.

#### Add Reader (implemented)

- **Actor(s):** User
- **Goal:** Registers the characteristics of a new Reader in the system.
- **Precondition:** The user is logged in as an administrator of the RFIDLocator. The following characteristics are available (M stands for mandatory):
  - Reader’s manufacturer.
  - Reader’s model.
  - LogicalAntennae information:
    - \* Location. (M)
  - PhysicalAntennae information:
    - \* EPC. (M)
    - \* Action (see Section 4.2). (M)
- **Workflow:** The user provides an XML file according to the reader’s configuration formalism and sends it to the application using the “Configure readers” page.
- **Postcondition:** The Reader characteristics are persisted or an exception is thrown and reported to the client.

### Remove Reader

- **Actor(s):** User
- **Goal:** Unregisters a Reader. If successful, information coming from this Reader is not going to be taken into account anymore.
- **Precondition:** The user is logged as an administrator of the RFIDLocator and the Reader was already added (see Add Reader).
- **Workflow:** Using the “Remove Reader” page, the user selects the RFID reader to be removed from the application.
- **Postcondition:** The Reader is removed or an exception is thrown and reported to the client.

### Modify Reader

- **Actor(s):** User
- **Goal:** Modifies the characteristics of a Reader (or of the antennae enclosed in a Reader). This use case is useful, for instance, when the Location of the PhysicalAntenna of a Reader is changing over the time (typically the case with a hand-held reader).
- **Precondition:** The User is logged as an administrator of the RFIDLocator and the Reader was already added (see Add Reader).
- **Workflow:** Using the “Edit Readers” page, the user changes the characteristics and configuration of 1..n readers.
- **Postcondition:** The new characteristics are persisted or an exception is thrown and reported to the client.

### Register User (implemented)

- **Actor(s):** User
- **Goal:** Registers a new User (administrator/employee) in the system. This is achieved in order to give him access to the RFIDLocator.
- **Precondition:** A non-existent username and additional information (e.g firstname, last-name, password, email, function, office location, degree of access to confidential information...) are provided in a valid format.
- **Workflow:** The administrator logs in to the application and moves to the “Add user” page. She provides various information about the new user and confirm the initialization of a new account.
- **Postcondition:** The User is correctly registered or an exception is throw and reported to the client.

### Unregister a User

- **Actor(s):** User
- **Goal:** Revokes access to the RFIDLocator for a given User. The User will be actually removed **only** when all the TraceableObjects she created (see Attach Tag) will be detached of their EPCs.
- **Precondition:** The administrator is logged. The username of the User to unregister is known.
- **Workflow:** The administrator logs in to the application and moves to the “Remove user” page. He then selects the user to remove and confirms the revocation.
- **Postcondition:** The access is revoked or an exception is thrown and reported to the client.

### Login (implemented)

- **Actor(s):** User
- **Goal:** Logs a RFIDLocator User in the system.
- **Precondition:** The User has previously been registered (see register User) and provides a valid username/password pair.
- **Workflow:** The user moves to the “Login” page and provides his user/password pair. If the provided information is correct he is then logged in the system for a given time.
- **Postcondition:** The User is correctly logged in or an exception is thrown and reported to the client.

### Logout (implemented)

- **Actor(s):** User
- **Goal:** Logs a RFIDLocator User out of the system.
- **Precondition:** The User was logged in.
- **Workflow:** The user clicks “Logout” and gets disconnected from the application.
- **Postcondition:** The User is correctly logged out or an exception is thrown and reported to the client.

### Log Observation (1) (implemented)

- **Actor(s):** Event Manager, RFID Reader
- **Goal:** Event Manager side of an observation’s report: Propagates an EPC observation made by a PhysicalAntenna.
- **Precondition:** The antenna which made the Observation was previously recorded in the system (see Add Reader).

The following information is available (M stands for mandatory):

- PhysicalAntenna’s EPC. (M)

It is worth noting that in the PML sent by the Event Manager, it will be named Reader’s EPC and not PhysicalAntenna’s EPC.

- Timestamp. (M)
- Detected EPC. (M)
- **Workflow:** The RFID reader catches the RFID events and reports them to the Event Manager. This latter filters the observations and sends them to a message queue.
- **Postcondition:** The Observation is sent or an error is logged by the Event Manager.

### Log Observation (2) (implemented)

- **Actor(s):** Application Server
- **Goal:** Application Server side of an Observation's report: Reads an Observation sent by the Event Manager and processes it.
- **Precondition:** The Observation arrived in a well formed JMS message. The message contains the following information (M stands for mandatory):
  - Timestamp. (M)
  - Detected EPC(s). (M)
  - PhysicalAntenna's EPC (named "Sensor"'s EPC in the PML message). (M)
- **Workflow:** Perform checks on the Observation:
  1. Check if the EPC of the PhysicalAntenna was registered in the RFIDLocator.
  2. Check if the TraceableObject is registered.
  3. Check if the read is valid (i.e. that the sequence of INs and OUTs matches the pattern given by the central algorithm).

If the three conditions are satisfied, persists the Observation for later use.

- **Postcondition:** The Observation is persisted or sent to a log (depending on the results of the above check-list).

### Configure the Readers (implemented)

- **Actor(s):** System administrator
- **Goal:** Configuration of an RFID Reader **on the Event Manager**.
- **Precondition:** The Reader to be configured has a valid EPC.  
The User is logged as the administrator of the Sun Java System RFID Software v.1.0.
- **Workflow:** The system (or the Administrator...) configures the RFIDConfig.xml of the Event Manager. The administrator manually restarts the EM.
- **Postcondition:** The system is correctly configured or an exception is logged by the Event Manager.

#### 4.3.2. Demonstration Use Cases

The use cases exposed below are not actually part of the RFIDLocator. However they were specified to emphasize the way of integrating the RFIDLocator to any other existent system.

### Create a Business Object

- **Actor(s):** Employee
- **Goal:** This use case is particular as it is not really part of this application. It consists of the creation of a Business Object, such as a medical file or a case in a law firm. After this first step, the newly created Business Object can then be registered as a TraceableObject of the RFIDLocator (see Attach Tag).
- **Precondition:** The data required to register the case are known.
- **Workflow:** The user logs in to his business system and creates a new Business Object in the pseudo business application.
- **Postcondition:** The Business Object is correctly created or an exception is thrown and reported to the client.

### Find a Business Object

- **Actor(s):** Employee
- **Goal:** When a Business Object was created using “Create a Business Object ”, it can be retrieved by providing the client’s name<sup>4</sup>.
- **Precondition:** The name of its related client (or a least part of it) is known.
- **Workflow:** The user logs in into his business system and attempts to look for a TraceableObject using a search string on a given page.
- **Postcondition:** The business number of the cases are returned or an exception is thrown and reported to the client.

---

<sup>4</sup>Client is to take in its vast sense, ranging from a patient to an enterprise...

# 5

## Software Architecture and Implementation

---

<b>5.1. Object Model of the RFIDLocator</b> . . . . .	<b>47</b>
5.1.1. Location . . . . .	47
5.1.2. PhysicalAntenna . . . . .	47
5.1.3. LogicalAntenna . . . . .	47
5.1.4. Reader . . . . .	50
5.1.5. Action . . . . .	50
5.1.6. TraceableObject . . . . .	50
5.1.7. User . . . . .	51
5.1.8. LocatorObservation . . . . .	51
5.1.9. BufferedObservation . . . . .	52
<b>5.2. The Services</b> . . . . .	<b>52</b>
5.2.1. UserManager . . . . .	52
5.2.2. LocationManager . . . . .	53
5.2.3. TraceableObjectManager . . . . .	53
5.2.4. ReaderManager . . . . .	53
5.2.5. PMLSimulatorPublisher . . . . .	55
5.2.6. ObservationManager . . . . .	56
<b>5.3. The SensorListenerMessageDrivenBean</b> . . . . .	<b>56</b>
<b>5.4. The Solvers</b> . . . . .	<b>56</b>
5.4.1. Introduction to the Solvers . . . . .	56
5.4.2. Architecture of the Solvers . . . . .	58
<b>5.5. Other Classes</b> . . . . .	<b>60</b>
<b>5.6. Summary of the Sequences</b> . . . . .	<b>60</b>
5.6.1. Solving Sequence . . . . .	60
5.6.2. Seeking a TraceableObject . . . . .	62
<b>5.7. Reader's Configuration Formalism</b> . . . . .	<b>62</b>

---

This chapters aims to summarize the architecture and the implementation of the application. It provides an overview of the Object Model, the algorithms and the special formalisms created for the RFIDLocator. It also offers a view of the programming techniques that were used for the concrete implementation.

As the application contains more than a hundred classes not all of them are described in this

chapter. However, the complete and commented API of the RFIDLocator is located on the CD-ROM of the application at: `api/`. The latest version of the API can also be accessed on the Internet from [3].

## 5.1. Object Model of the RFIDLocator

**Package:** `ch.unifr.diuf.RFIDLocator.ejb.entity`

The core of the RFIDLocator is built around a relatively small set of objects as shown on Figure 5.1. All the objects on this figure are modeled using Entity Beans. An Entity Bean (aka Entity) is an abstraction of data in a storage medium. In the case of the RFIDLocator, the Entities represent the tuples in a relational database. The place of the Entity Beans among the other components of the Enterprise JavaBeans is summarized on Figure 5.2. Implementing the Objects of the model using Entity Beans is not mandatory. The reasons for this choice are exposed in Chapter 6.

The role of each Entity is a central point for understanding the structure of the RFIDLocator. As a consequence we will briefly describe the role of each of the Object in the model. Note that some of these were already defined in Section 4.2 in the use cases' perspective. Some are defined here once again in the perspective of the application's architecture.

### 5.1.1. Location

The Location object models a "place" within the area controlled by the RFIDLocator. It is composed of two fields, a description and a `businessLocationNumber`. The former is a non-formalized way of describing the place in question whereas the latter is a formalized (enterprise-wide) description. For instance: RM 3-3.113 is the formalized (university-wide) description of "The office of Dr. Patrik Fuhrer".

The Location is the central object of the RFIDLocator as this latter aims to **locate** physical objects. That is, to answer to the question: given a physical object to look for, to which registered Location did it go through. As a consequence the `LocatorObservations` (i.e. observations that are valid in the context of the RFIDLocator) are always linked to a Location.

### 5.1.2. PhysicalAntenna

A `PhysicalAntenna` represents a hardware component able to capture RFID events by emitting an electro magnetic field. A `PhysicalAntenna` has a field called `ecpString` used to uniquely identify it. This field is meant to contain an EPC in the form of a pure entity URI (see Subsection 3.2.1). However, just about any unique identifier that can be expressed as a string will fit the application's needs.

Finally, in the object model, a `PhysicalAntenna` is always connected to a `LogicalAntenna`.

### 5.1.3. LogicalAntenna

A `LogicalAntenna` is an aggregate of  $1..n$  `PhysicalAntennae`. It is the central component in the solving of an Observation. A `LogicalAntenna` is composed of many fields and offers various methods as shown on Figure 5.3. Perhaps the most important among these is the `jndiNameConcreteObservationSolver`. It contains the JNDI (ENC) name of the Solver that the `LogicalAntenna` will be using to decide what to do with a given Observation. Bascially, the concept of `LogicalAntenna` was created in order to be able to capture the direction of the motion using  $2..n$  `PhysicalAntenna`. Indeed, most RFID readers do not have the native ability to capture the direction. This topic is further described in Section 5.4.

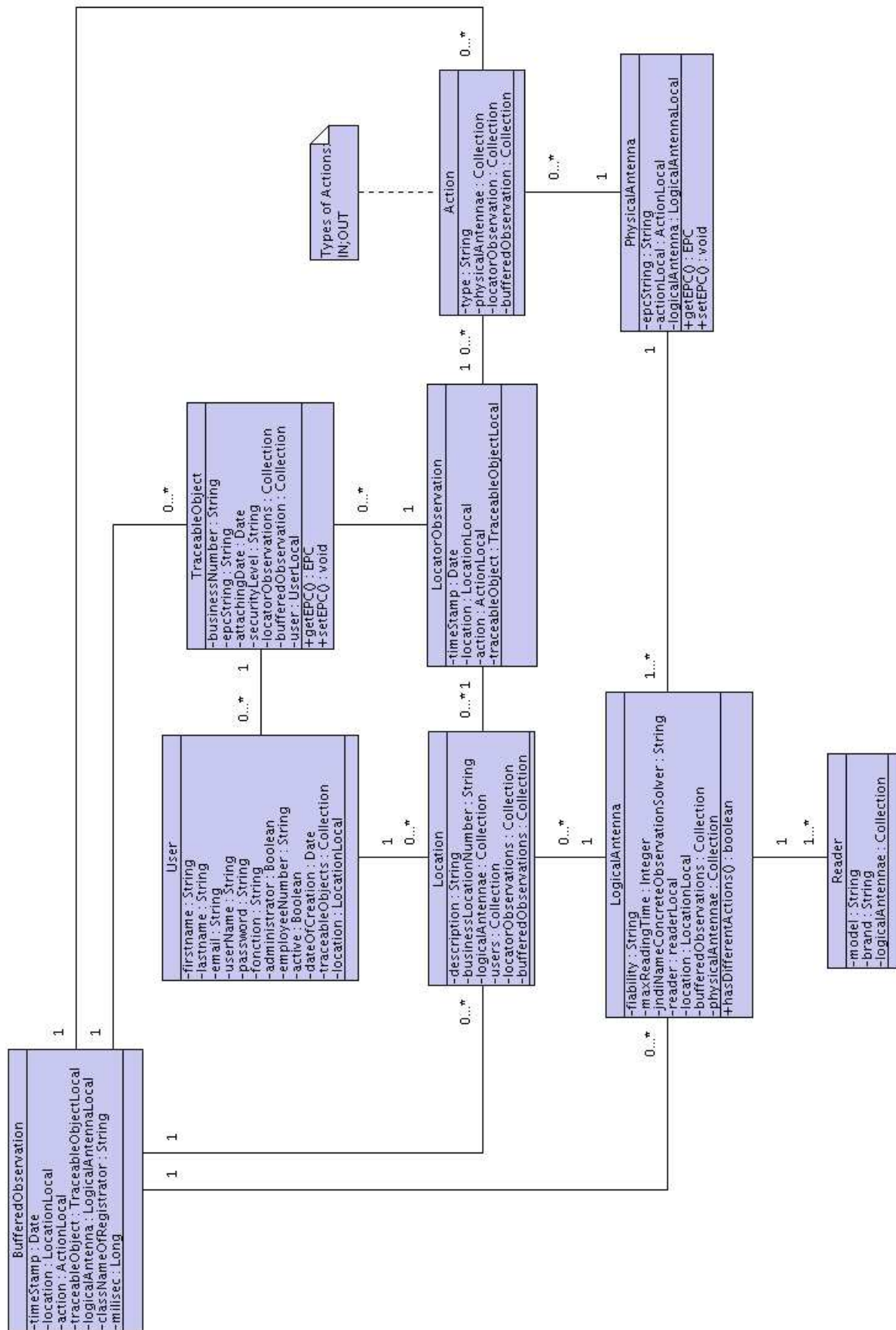


Figure 5.1.: The Object Model of the RFIDLocator

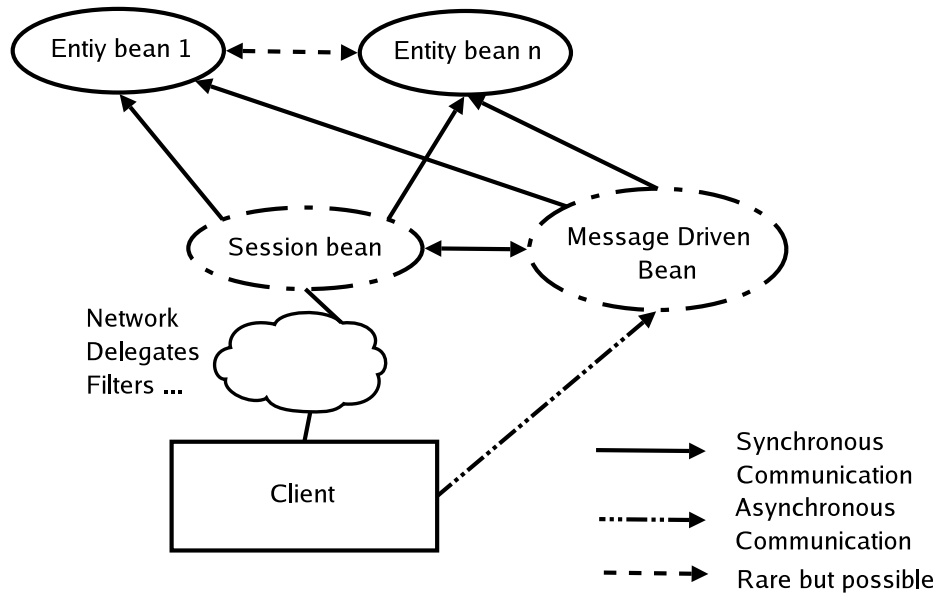


Figure 5.2.: Elements of the EJB framework

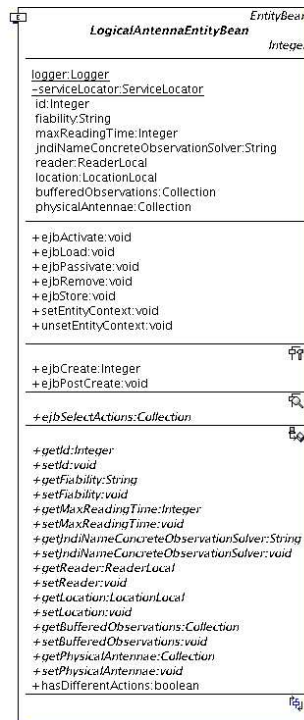


Figure 5.3.: Class diagram of a LogicalAntenna

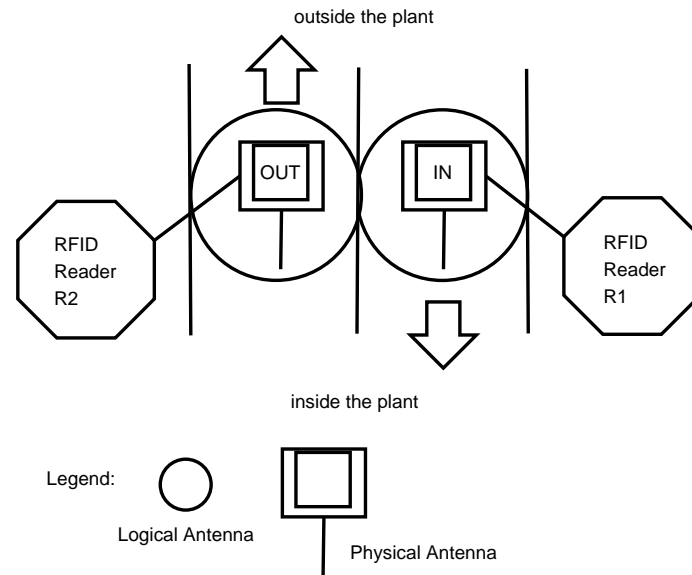


Figure 5.4.: Assigning actions for a gate

#### 5.1.4. Reader

The next object represents a Reader which is a device that commands 1.. $n$  PhysicalAntennae in order to detect tagged objects within their fields. The Reader object does not contain much information. It is primarily intended to group a number of LogicalAntenna together which could be of great help in some circumstances (e.g. when removing a reader from the system).

#### 5.1.5. Action

An Action is the fundamental element for the algorithm that traces physical objects. This version of the RFIDLocator supports two actions: IN and OUT. A IN on a Location means that the object entered the Location, whereas an OUT means that the object exited the Location. The INs and OUTs are distinguished at the PhysicalAntenna level. That is: a PhysicalAntenna is always attached to either an IN or an OUT Action. The way the algorithms use these elements to find the actual Action that happened with an Observation is described in Section 5.4.

To emphasize the use of Actions let us consider an example:

Two RFID readers  $R_1, R_2$ , are placed at the gate of a plant.  $R_1$  at the entrance,  $R_2$  at the exit. Hence, the PhysicalAntennae of  $R_1$  should be assigned IN Actions saying that objects detected by these antennae are going **in** the plant. On the other side, the PhysicalAntennae of  $R_2$  will be assigned OUT Actions reporting that the detected objects are going **out** the plant. Figure 5.4 summarizes the setting. At this point it is worth noting that a LogicalAntenna might be able to report both IN and OUT Actions. This special case is exposed in Section 5.4 as well.

The syntax to assign a PhysicalAntenna a particular Action is described in Section 5.7. Finally let us provide a concrete example: Figure 5.5 presents the use of a LogicalAntenna composed of one PhysicalAntenna that can report only IN Actions. If you put such a setting on a desk in a office, it will report all the documents that are currently IN this office.

#### 5.1.6. TraceableObject

Such an object is a physical element that can be tracked by the RFIDLocator. It could be about anything that is physically big enough to hold an RFID tag. The TraceableObject is an interface between the legacy business system and the RFIDLocator. As a consequence, it contains two fields that uniquely identify the object:



Figure 5.5.: A concrete example using a IN Action

- One on the business system side: the `businessNumber` (e.g. Case-Veronique-Gu-2411).
- One on the RFIDLocator side: the `epcString` (e.g. `urn:epc:id:gid-96:1.1.150`).

The use of both these unique identifiers enables an object to be searched by `businessNumber` even if this latter is actually identified by its `epcString` by the RFIDLocator and attached readers. A `TraceableObject` is always attached to the `User` who registered it. Depending on the policies of the organization running the RFIDLocator, displaying it when querying for the location of an object could be of great help. Indeed, in some cases, this `User` is likely to have a better idea of where the `TraceableObject` might be located.

### 5.1.7. User

A `User` is basically someone allowed to access and query the RFIDLocator. This version of the application does not distinguish the users according to their respective rights (administration, querying, etc.). Yet the reader should note that this fact could be of great importance in a commercial application.

Some information (name, email, etc.) about the employee in question are retained. It is worth noting that this could introduce redundancy in the case a user database already exists (which is often the case). These fields were added for demonstration purpose, a better approach is certainly to build an interface between the existing user (employee) database and the RFIDLocator.

### 5.1.8. LocatorObservation

It is the persistent result of an `Observation`. Its semantics is basically that a `TraceableObject` was **observed** at a given time going IN or OUT a particular `Location`. When tracing `TraceableObject`s the queries will be made on these objects.

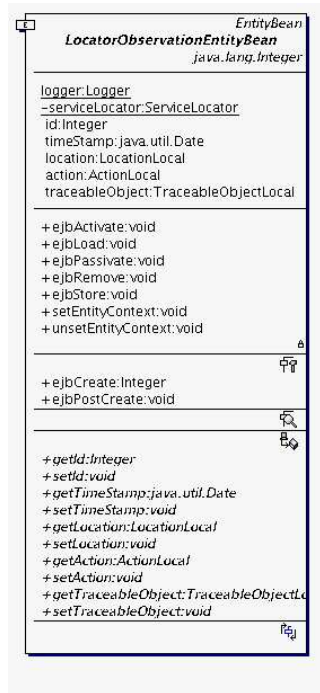


Figure 5.6.: Class diagram of a LocatorObservation

### 5.1.9. BufferedObservation

Such an Observation is basically a potential LocatorObservation. It is used by the solving algorithms when elements are missing to actually persist the Observation.

Unlike a LocatorObservation each of these objects is connected to a LogicalAntenna. This can be explained by the fact that the algorithms solving the Observations are commanded by the LogicalAntennae.

## 5.2. The Services

**Package:** `ch.unifr.diuf.RFIDLocator.ejb.session`

Where Entity Beans are used to represent actual data in a storage medium, Session Beans (aka Sessions) contain the specific business logic of the application. Thus, the RFIDLocator proposes several Session Beans offering business services. The place of the Session Beans in the overall EJB framework is shown on Figure 5.2. Basically, Session Beans (or Sessions) are accessed by the client (often through a number of interfaces and proxies) to solve a “business task”. In turn, the Sessions are interacting with 1..*n* Entities to reach the goal.

This section provides an overview of **the most important SessionBeans** as well as a summary of the service they offer.

It is worth noting that most of the Session Beans described in this section implement the Session Façade Pattern (see [ACM03]). That is: where the client could access the Entities directly, the Façades provide the client with a unique entry point to gain access to many Entities at the same time. Figure 5.7 presents a typical Session Façade.

### 5.2.1. UserManager

The UserManagerSessionBean offers methods related to the management of RFIDLocator’s Users. As an example, the method `registerUser()` can be used to add a new User to the system.

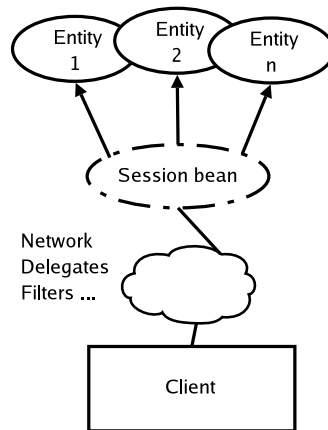


Figure 5.7.: A typical Session Façade

### 5.2.2. LocationManager

This Session is intended to offer methods regarding the Locations. For instance, the method `addLocation()` provides a way of creating a new Location. The newly created Location is then going to be part of the places the `RFIDLocator` can monitor (provided a `PhysicalAntenna` is placed in this Location). It is primarily intended to be used by other Session Beans (such as the `Sensor Manager Session Bean`)

### 5.2.3. TraceableObjectManager

The `TraceableObjectManager` offers methods for managing the objects that can be traced by the `RFIDLocator`, i.e. the `TraceableObjects`. But it also provides a central method called `locationHistory()` which is in charge of returning the Locations a `TraceableObject` went through. This latter service is the core business of the `RFIDLocator` as it permits the approximating of the current place an object is in.

### 5.2.4. ReaderManager

This Session basically provides a method for parsing an XML file containing the settings of the readers to install in the `RFIDLocator`'s environment. Indeed, the method `ParseConfigString()` takes an XML string as argument and builds an object graph containing the following objects:

- Readers
- LogicalAntennae
- PhysicalAntennae
- Solvers
- Locations

It does so by using Sun's implementation<sup>1</sup> of the Java Architecture for XML (Data) Binding (JAXB). This specification enables the conversion of an XML document into Java Objects (called Content Object in this context) in a very straightforward and elegant manner.

As a proof consider the code extract on Figure 5.10. It was taken from the `ReaderManagerSessionBean`. As it can be seen on this figure, JAXB enables the programmer to translate XML

<sup>1</sup>Sun's implementation of JAXB is bundled into the Java Web Services Developer Pack 1.5 (jwsdp-1.5), available for download at [15].

---

```
<RFIDLocator>
  <Reader>
    ...
  </Reader>
  ...
</RFIDLocator>
```

---

Figure 5.8.: A very simple XML code

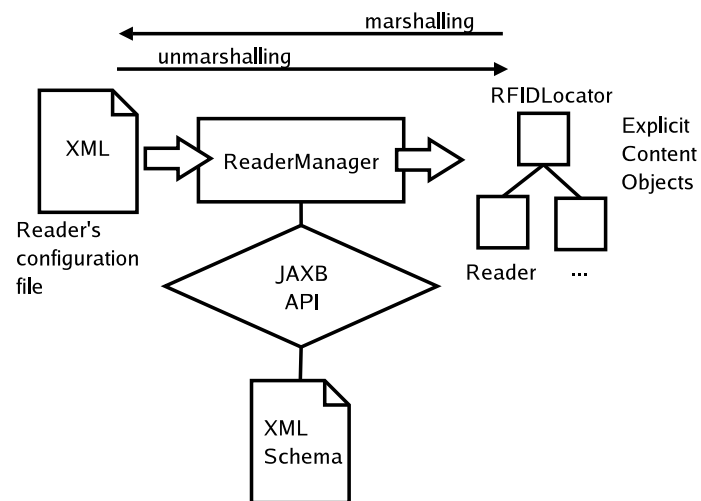


Figure 5.9.: The marshalling/unmarshalling process using JAXB

---

```

public void parseConfigString(String configString)
throws JAXBException,
    IOException, ServiceLocatorException, CreateException,
    FinderException{

    //use the classes generated by JAXB (from the XML-Schema)
    //as the new context of the JAXB marshaller/unmarshaller
    JAXBContext jc = JAXBContext.newInstance
        ("ch.unifr.diuf.RFIDLocator.config");

    File configFile = new File("./RFIDLocatorConfig.xml");
    PrintWriter outFile = new PrintWriter
        (new FileWriter(configFile));
    outFile.print(configString);
    outFile.close();

    //create an unmarshaller (XML->Content Objects) object
    Unmarshaller unmarshaller = jc.createUnmarshaller();
    unmarshaller.setValidating(true);

    //unmarshall the XML into Content Objects
    RFIDLocator rfidLocator = (RFIDLocator)
    unmarshaller.unmarshal(configFile);

    List readers = rfidLocator.getReader();
    Iterator readersIterator = readers.iterator();

    [...]

```

---

Figure 5.10.: The JAXB unmarshalling of a XML string

codes into Content Objects within just a few lines. The newly created objects and their content can then be accessed explicitly. For instance, consider the XML code on Figure 5.8. Accessing the Reader element can be done by writing: `RFIDLocator.getReader()`. This is more elegant and explicit than the JDOM approach for instance, where accessing Reader would require the following: `Root.getChild()`.

However, it is worth noting that to be able to marshal an XML code into Content Objects, one first needs to generate the binding classes using a tool provided by every JAXB vendor.

Figure 5.9 provides a graphical view of the unmarshalling process used by the ReaderManager. Eventually, after creating the Content Objects the ReaderManager persists them into the corresponding Entity Beans as described in the Object Model (see Section 5.1).

### 5.2.5. PMLSimulatorPublisher

In order to test the system without the need of many (or even one) RFID readers, the RFIDLocator is provided with a PMLSimulatorPublisher. This Session offers methods for simulating the sending of Java Message Service (JMS) PML events (see Section 3.3) to the Message Driven Bean of the application (see Section 5.3). For sending such messages it provides two overloaded

methods: `publishPML(String PMLCoreString)` and `publishPML()`. The latter offers to send a static message whereas the former proposes to transport a customized message.

### 5.2.6. ObservationManager

The `ObservationManager` is in charge of persisting `Observations` through the method `addLocatorObservation()`. Note that an `Observation` has to go through various steps before being eventually persisted as a `LocatorObservation`. These steps are the matter of Subsection 5.4.

## 5.3. The *SensorListenerMessageDrivenBean*

**Package:** `ch.unifr.diuf.RFIDLocator.ejb.mdb`

The `SensorListener` is the integration point between the Event Manager and the `RFIDLocator` Server on which the `RFIDLocator` is installed. As described in Section 6.3 the events reported by the Event Manager go through several steps ending on a JMS Queue called the `RFID Locator Queue`. The `SensorListener` is a Message Driven Bean (MDB) listening to this latter Queue. That is, the `SensorListener` is a component being awakened by the EJB container each time a message is posted on the `RFID Locator Queue`.

A message arrives at the MDB in the form of a PML string. The PML is then unmarshalled using the JAXB API just as described in Section 5.2.4. After this conversion, the Message Driven Bean does the first filtering by checking whether the `PhysicalAntenna` that made the `Observation` is registered withing the `RFIDLocator`. If this turns out to be the case, the `SensorListener` contacts the corresponding `LogicalAntenna` (or its proxy to be more precise) and asks it to “solve” the `Observation`. That is: to decide what to do with the incoming `Observations`.

Such a Message Driven Bean is a very convenient way of listening to a Queue. Indeed, the EJB container automates all the receiving work which enables us to concentrate on the business logic of the asynchronous component.

The concrete role of the MDB is shown on Figure 5.14. The `SensorListener` is in charge of the steps 1.1 to 1.4 depicted on this figure.

## 5.4. The Solvers

### 5.4.1. Introduction to the Solvers

**Package:** `ch.unifr.diuf.RFIDLocator.ejb.solver`

The Solvers are Session Beans as well. In short, the Solvers are the **algorithms** used by the `RFIDLocator` to trace the position of the `TraceableObjects`. As said above when a message arrives on the Queue of the application the MDB contacts the corresponding `LogicalAntenna` and asks it to solve the incoming `Observations`. To do so, the `LogicalAntenna` passes the `Observations` to its **Solver**. According to its algorithm a Solver has three possibilities when handling of an `Observation`:

1. Persist the `Observation` as a `LocatorObservation`, which can be interpreted as a direct validation of the incoming `Observation`.
2. Buffer the `Observation` as a `BufferedObservation` in order to wait some more information before actually taking a decision.
3. Discard the `Observation`, which can be interpreted as declaring it to be invalid.

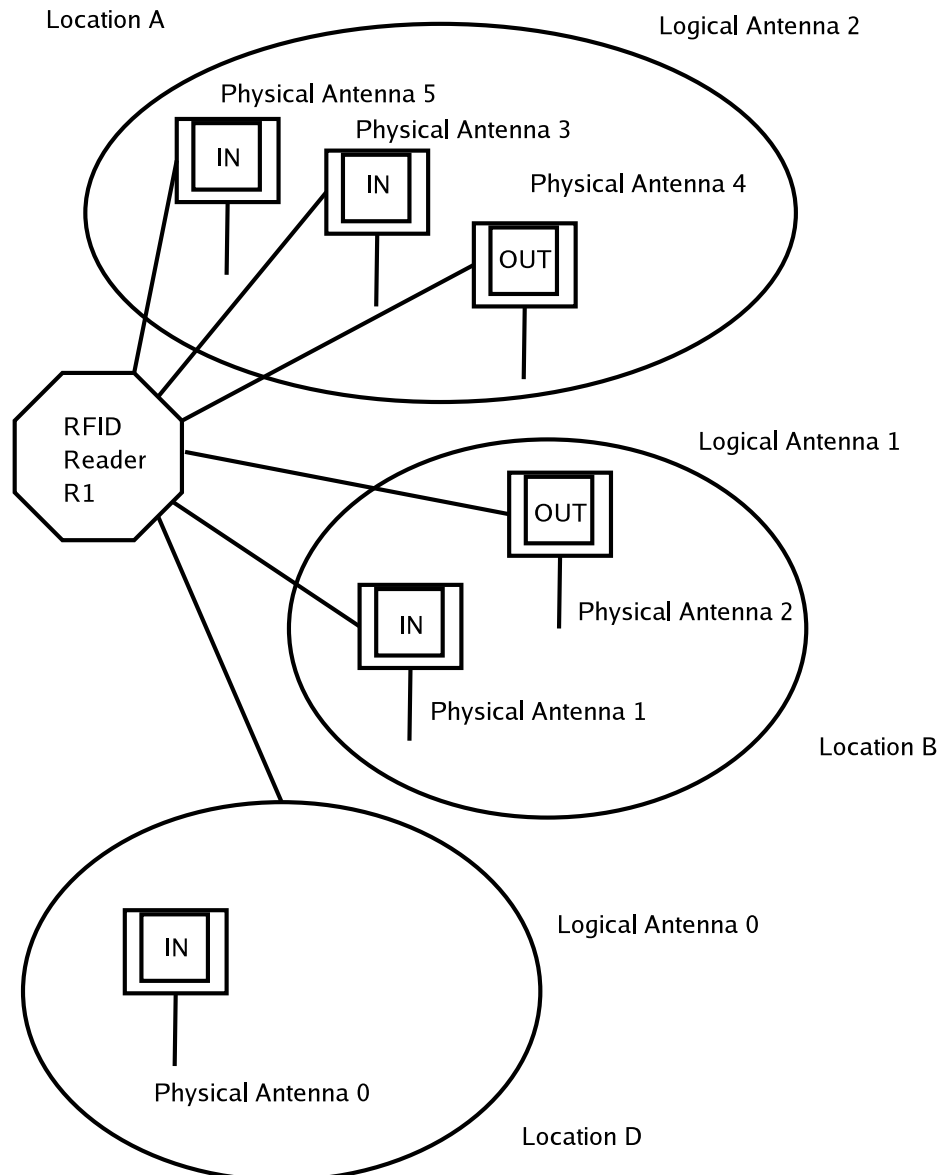


Figure 5.11.: A typical RFIDLocator reader's setting

**How** the decision is made is the business of the algorithm implemented in the Solver. The idea behind this concept is quite simple: there are many ways of identifying objects and thus, the system should support an unlimited number of algorithms to validate (or “solve”) Observations. To emphasize the differences between the Solvers let us take some examples. Consider the reader  $R_1$  of Figure 5.11. It shows a reader to which are attached three LogicalAntennae. In turn, the LogicalAntennae contain up to three PhysicalAntennae. These are each assigned an Action as explained in Section 4.1.

From the figure we can see that the Observation solving occurs at a LogicalAntenna level. LogicalAntenna 1 controls the Observations for Location B, LogicalAntenna 2 for Location A and so on. To begin with let us focus on the solving of an Observation that occurred at PhysicalAntenna 0. From the fact that the LogicalAntenna 0 contains only one PhysicalAntenna and thus can report only one type of Action it follows the simple validation rule:

- As long as the TraceableObject observed at PhysicalAntenna 0 is registered in the RFIDLocator, validate the Observation (i.e. persist it as a LocatorObservation).

This rule is the core of the Solver for LogicalAntenna 0. It has been implemented in this version of the RFIDLocator and is called: Single Type Concrete Solver.

The case of the LogicalAntennae 1 and 2 needs more attention. Indeed, in both settings the LogicalAntennae can report IN **and** OUT Actions. The use of such antennae is not obvious at first. One could, for instance, think of separating the two PhysicalAntennae of the LogicalAntenna 1 into two distinct LogicalAntennae. As a result the Single Type Concrete Solver would do the solving job perfectly. However, coupling both PhysicalAntennae and creating a new Solver enables to **capture the direction of the motion** without the need of a very sophisticated reader.

Indeed, the algorithm of LogicalAntennae 1 and 2 must only follow this set of rules; for each incoming Observation:

1. Check if the TraceableObject is registered, if not: discard the Observation and quit the algorithm.
2. Scan the BufferedObservations to find at least one Observation:
  - a) With the **opposite** Action ( $IN \rightarrow OUT$  or  $OUT \rightarrow IN$ ).
  - b) Which has not expired in regards to the MaxReadingTime attribute of the LogicalAntenna.

If none is found goto 3, else goto 4.

3. Buffer the Observation as a BufferedObservation and quit the algorithm.
4. Persist the Observation as a LocatorObservation. The Action attached to this latter is going to be the last observed Action.

Additionally, the buffer is cleaned before and after the solving, deleting BufferedObservations that are too old to be part of a valid motion. Hence, using such a Solver both LogicalAntennae 1 and 2 can capture the direction of the motion. This algorithm is implemented in the RFIDLocator. It is called the Different Actions Concrete Solver.

To emphasize the idea of capturing motion consider the concrete example of Figure 5.12. On this screenshot you can see two PhysicalAntennae. Using a configuration file (see Section 5.7), both are aggregated forming a single LogicalAntenna. Now, using the Different Actions Concrete Solver as described above, these two PhysicalAntennae can report whether an object was going OUT of the office or IN the office.

### 5.4.2. Architecture of the Solvers

**Package:** ch.unifr.diuf.RFIDLocator.ejb.solver

As seen in the previous section Solvers implement algorithms used to validate an Observation. Thus, the number of possible Solver is unbound. To satisfy this criterion the Solvers are based on a modular architecture.

To start with, all the Solvers must implement the Observation Solver interface. It defines a single method called solve. This method is called by the LogicalAntennaSolvingProxy when the validation of an Observation is required. As the solving is done at the level of the LogicalAntennae, these maintain a reference to their respective Solver. For instance, the LogicalAntenna 0 of Figure 5.11 has a reference to the Single Type Concrete Solver.

Thus, to add a new Solver to the RFIDLocator one just need to program it and to have the concerned LogicalAntennae referencing it.

Two Solvers were included with this version of the RFIDLocator. As shown on Figure 5.13 both of them are implemented as Session Beans because of the high performance of these components. However, this is not mandatory and a Solver could as well be a POJO implementing



Figure 5.12.: A concrete example of capturing the direction of the motion

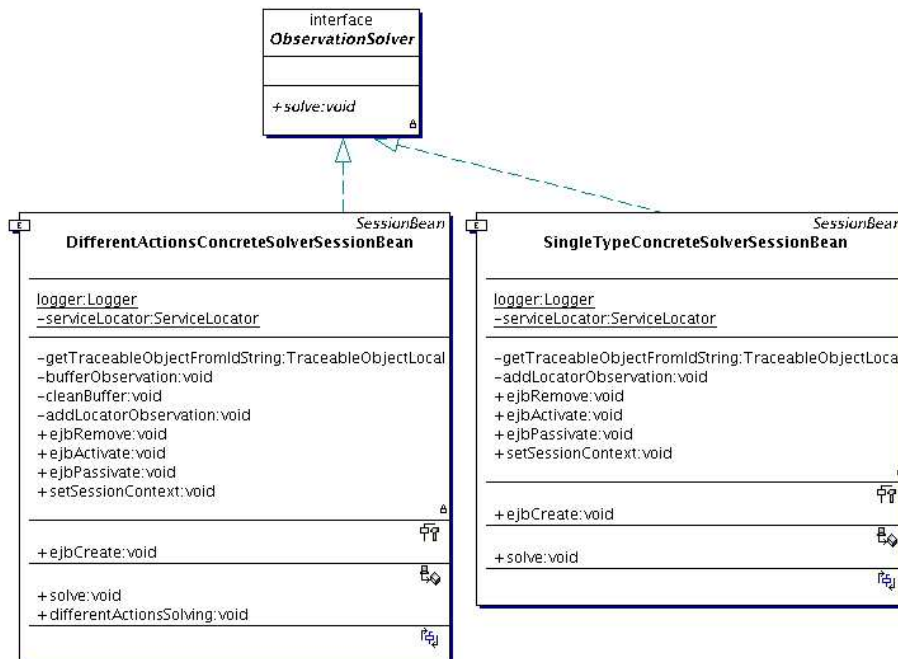


Figure 5.13.: The Solvers provided with the RFIDLocator

the Observation Solver interface.

Eventually it is worth noting that some classes are intended to facilitate the writing of new Solvers. These are located in the `ch.unifr.diuf.RFIDLocator.ejb.session` package. The `BufferedObservationManager` offers a complete set of methods for buffering Observation and managing the buffered entries. Additionally, the `ObservationManager` proposes methods for persisting Observations once these are valid.

## 5.5. Other Classes

Many other classes are part of the `RFIDLocator`. Some are intended for modeling useful Objects (like the EPC classes located in `ch.unifr.diuf.RFIDLocator.ejb.util`), others are implementing J2EE design patterns (see [ACM03]) to improve the design of the application. Among the patterns implemented in the `RFIDLocator` we find:

- Transfer Objects (TOs) (see package: `ch.unifr.diuf.RFIDLocator.to`) and Composite Transfer Objects: used to transport objects across tiers.
- Session Façades (see `ch.unifr.diuf.RFIDLocator.session`) that permit the access of various Entities through a single object.
- Business Delegates (see package: `ch.unifr.diuf.RFIDLocator.ejb.delegate`): meant to hide the complexity of remote communications to the client.
- A Service Locator (see package: `ch.unifr.diuf.RFIDLocator.ejb.util.serviceLocator`) used to abstract the request of objects through JNDI (or JNDI ENC).
- A Sequencer (see package: `ch.unifr.diuf.RFIDLocator.ejb.util.sequence`) that is in charge of managing the primary keys for the entity beans. The use of such a sequencer to assign keys reduces the risk of incompatibilities between the different databases and Application Servers. For more information on this particular topic please refer to [McL02].

## 5.6. Summary of the Sequences

This section presents the two most important sequences of operations for this application. It gives an overview of the components interacting together for:

1. Solving an Observation
2. Querying the `RFIDLocator` to locate a `TraceableObject`

However, for the sake of simplicity, the diagrams presented in this section omit some steps that are not mandatory for understanding the architecture of the application. These omitted steps include for instance, some calls to the Business Delegates, the Sequencer or the Service Locator. It also neglects the calls to the interfaces (Local, Remote, Home, etc.) required by the EJB specification.

### 5.6.1. Solving Sequence

The core of the `RFIDLocator` is certainly the ability to convert RFID (EPC) Events (i.e. Observations) into `LocatorObservations`. Figure 5.14 presents the simplified sequence diagram activated when an Observation is posted on the Queue of the application.

As exposed in Section 6.3, the message originates from the Event Manager in the form of a PML stream. The `SensorListener` first checks whether the `PhysicalAntenna` that made this Observation

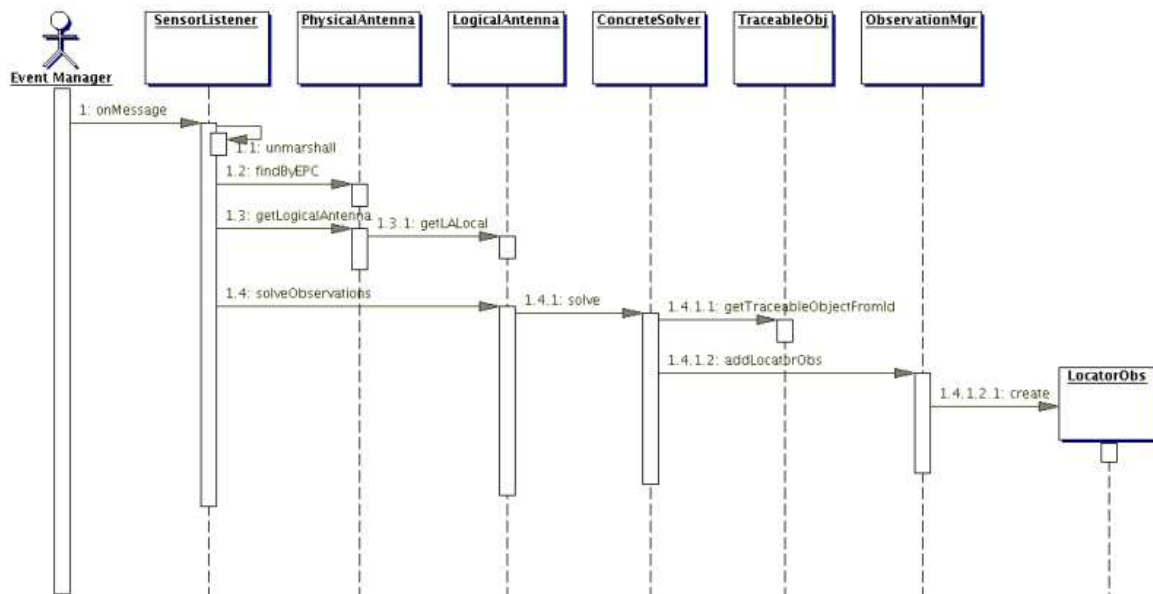


Figure 5.14.: Simplified sequence diagram of an Observation solving

is registered within the RFIDLocator. If so, it queries for the LogicalAntenna the PhysicalAntenna belongs to. Because the relations between the Entities are coded in the Deployment Descriptors of the application (thanks to the EJB specification) this operation is really straightforward. A call to `getLogicalAntenna()` on the PhysicalAntenna will immediately return the enclosing LogicalAntenna. Thus, one should note that the call to `1.3.1: getLALocal` is never really issued by the code of the RFIDLocator. Indeed, this call is managed by the EJB container. It was included here only to emphasize that the corresponding LogicalAntenna is returned (and activated) after this automated step.

The SensorListener then executes the `solveObservation()` method on the LogicalAntenna. Abstracting the code what it does at this step is simply asking the antenna to achieve the rest of the job. It is worth noting that, in fact, the SensorListener calls a proxy of the concerned LogicalAntenna (called the LogicalAntennaSolvingProxy included in package: `ch.unifr.diuf.RFIDLocator.session`). Even if this option is less clear in terms of design it was the only possible solution. A direct call to the LogicalAntenna blocks the component (because the transactions are set to Required, see [MH04])<sup>2</sup>. Since the LogicalAntenna is required again at various steps of the solving process, “Entity already in a transaction” exceptions are raised. Because it frees the Entity, the use of a “proxy” solves this issue without involving too much overhead.

The ConcreteSolver of the antenna is then called. One should note that for the example of Figure 5.14 a Single Type Concrete Solver was chosen. The fundamental idea for the Different Actions Concrete Solver is the same. However, since this latter solver also interacts with a buffer of Observations it requires slightly more steps (see Subsection 5.4).

The Solver first checks whether the observed objects are TraceableObjects if not, the solving ends without persisting anything. On the contrary if TraceableObjects are detected, the Solver asks the ObservationManager to persist the events as LocatorObservations. This is done by creating a LocatorObservation Entity Bean, whose data is then written to a relational database.

The Observations are valid and the solving process is over.

<sup>2</sup>Note that setting the transactions for the Entity to `Require New` solves this issue. However many other problems appear with this solution, most of them regarding the creation of new Entities...

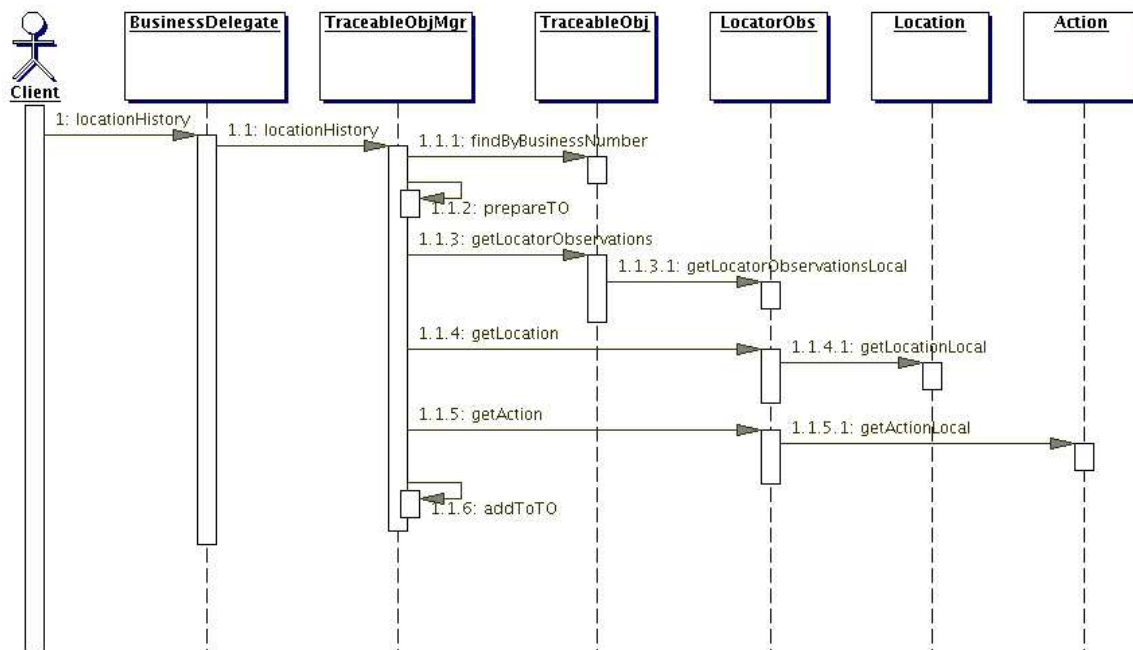


Figure 5.15.: Seeking a TraceableObject: simplified sequence diagram

### 5.6.2. Seeking a TraceableObject

The next important sequence of operations is the seeking of a TraceableObject. In fact, this sequence is the main service the RFIDLocator offers to the end-user. It permits to get all the Locations a TraceableObject actually went through.

Figure 5.15 presents a sequence diagram of the operations. The client first contacts the corresponding Business Delegate passing it the Business Number of the TraceableObject to look for. This class, which abstracts the overhead of the remote communications, contacts the TraceableObjectManager on the remote host. Using the given Business Number the Manager issues a `findByBusinessNumber()`. The corresponding TraceableObject is returned. The manager can now start to build the Transfer Object containing the results. It first queries for the LocatorObservation corresponding to the TraceableObject (`getLocatorObservation()`). Having the Observation it now needs the Location where it was made. A simple `getLocation()` on the LocatorObservation solves the issue. Finally, the Action corresponding to the Observation is needed. Thus, `getAction()` is called on the LocatorObservation.

As explained above, the calls 1.1.3.1, 1.1.4.1 and 1.1.5.1 are never actually written by the programmer. Indeed, the relations between the Beans of the Object Model (see Section 5.1) are described in the deployment descriptors generated by XDoclet. As a consequence the EJB container automates the retrieving of objects that are part of a relation. For instance a `getAction()` call on a LocatorObservation will automatically return the actual Action corresponding to the LocatorObservation we are using.

## 5.7. Reader's Configuration Formalism

As seen before, the Object Model of the RFIDLocator has quite a number of elements. Thus, to configure the application in a real environment (or even in a test environment) one needs to be able to create and parametrize the objects composing it. Telling which PhysicalAntenna is part of which LogicalAntenna, what Location is controlled by which Reader and so on.

There are plenty of possibilities to achieve this goal. One is to program a fully-featured GUI

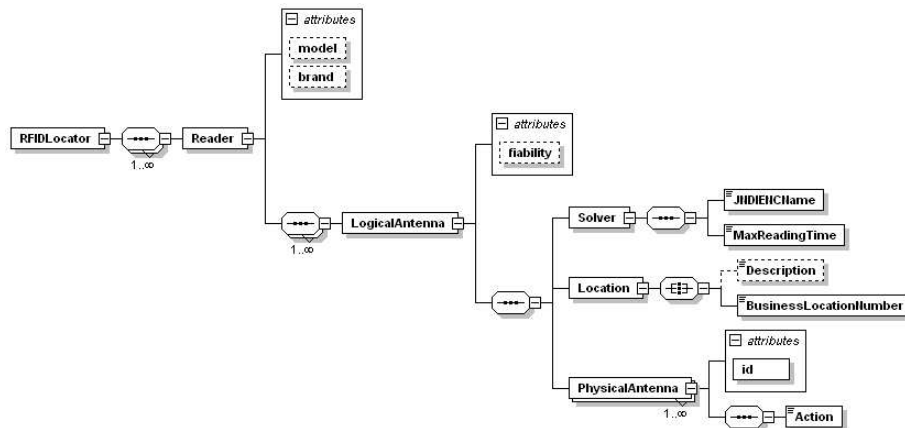


Figure 5.16.: Diagram of the Reader's Configuration Formalism

(Graphical User Interface) which is a good idea for the user but not so optimal in terms of flexibility and extension. Another would be to develop a special language to describe the configuration of the environment, which is against a standardized architecture. A third solution is to create an XML based formalism to describe and validate the environment of the RFIDLocator. The latter approach is the one implemented in the application.

The “Reader's Configuration Formalism” is based on a XML Schema created for the RFIDLocator. A copy of this schema (`RFIDLocator_v_0.xsd`) is provided in Appendix B.2.

To better understand the constraints and the use of this schema let us provide an example. Figure 5.17 fully describes a single RFID reader. A Reader is composed of at least one LogicalAntenna. In turn, a LogicalAntenna must contain:

- The JNDI ENC name of a Solver.
- A Location, which is nothing but the place in the environment it controls.
- $1..n$  PhysicalAntenna(e) which are the “eyes” of the Reader.

The Reader described by the file of Figure 5.17 is composed of two LogicalAntennae: one of them is able to track TraceableObjects in room RM-S-3.113, the other in room RM-S-3.114. Additionally, the first LogicalAntenna can capture the direction of the motion (i.e. can report two different Actions, see 5.4) whereas the second LogicalAntenna can only report IN events.

The hierarchy of the elements in the Reader's Configuration Formalism is summarized on the diagram of Figure 5.16.

As explained in Section 5.2.4, the configuration file is parsed (unmarshalled to be more precise) by the ReaderManager. In order to ensure the integrity of the configuration, this latter component is also responsible for checking the validity of the input stream against the XML schema before processing to the conversion into RFIDLocator Objects.

---

```

<?xml version="1.0" encoding="UTF-8"?>
<RFIDLocator xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="RFIDLocator_v_0.xsd">
  <Reader brand="TagSys" model="Medio L100">
    <LogicalAntenna fiability="80">
      <Solver>
        <JNDIENCName>
          ejb/DifferentActionsConcreteSolver
        </JNDIENCName>
        <MaxReadingTime>5000</MaxReadingTime>
      </Solver>
      <Location>
        <BusinessLocationNumber>
          RM-S-3.113
        </BusinessLocationNumber>
        <Description>
          The office of P.Fuhrer and G.Mostefaoui
        </Description>
      </Location>
      <PhysicalAntenna id="urn:epc:id:gid:25.1.10">
        <Action>IN</Action>
      </PhysicalAntenna>
      <PhysicalAntenna id="urn:epc:id:gid:25.1.11">
        <Action>IN</Action>
      </PhysicalAntenna>
      <PhysicalAntenna id="urn:epc:id:gid:25.1.12">
        <Action>OUT</Action>
      </PhysicalAntenna>
    </LogicalAntenna>
    <LogicalAntenna fiability="98">
      <Solver>
        <JNDIENCName>
          ejb/SingleTypeConcreteSolver
        </JNDIENCName>
        <MaxReadingTime>0</MaxReadingTime>
      </Solver>
      <Location>
        <BusinessLocationNumber>
          RM-S-3.114
        </BusinessLocationNumber>
        <Description>
          The office of Pr. Dr. J.Pasquier
        </Description>
      </Location>
      <PhysicalAntenna id="urn:epc:id:gid:25.1.13">
        <Action>IN</Action>
      </PhysicalAntenna>
    </LogicalAntenna>
  </Reader>
</RFIDLocator>

```

---

Figure 5.17.: XML document based on the RFIDLocator\_v\_0.xsd schema

# 6

## Hardware and Software Choices

---

<b>6.1. Software Choices</b> . . . . .	<b>65</b>
6.1.1. The Implementation of the Event Manager . . . . .	65
6.1.2. Enterprise JavaBeans . . . . .	66
6.1.3. XDoclet . . . . .	67
6.1.4. Application Server Software . . . . .	67
<b>6.2. Hardware Settings</b> . . . . .	<b>69</b>
6.2.1. Readers and Connectors . . . . .	69
6.2.2. The Server of the Event Manager . . . . .	69
6.2.3. The Machine of the Application Server . . . . .	71
6.2.4. RFID Labels . . . . .	71
<b>6.3. Summary</b> . . . . .	<b>73</b>

---

From the definition of its basis it should be clear that the RFIDLocator is a good candidate for a distributed application. Indeed, the need to be able to use it from anywhere and not just on the computer which processes the observations, is a reality. Having this fact in mind, the following sections present both the software and hardware technology choices that were made for the application.

### 6.1. Software Choices

#### 6.1.1. The Implementation of the Event Manager

The first software component required for an application working with RFID and EPCs is an implementation of the Event Manager. Many such software are sold on the market but not all of them meet the Savants' standards elaborated by the EPCglobal. As this work focused on an evaluation of the Sun Java System RFID Software v.1.0 the choice was quite easy to make. However, Sun's implementation of the Event Manager is not the only one to fit into the standards. Among many of them let us cite the RFID solution proposed by IBM (see [Col04] and [Sta]) or the RFID software proposed by SAP as an extension of its well known ERP.

It is still worth noting that the RFID software from Sun was one of the first Savant's implementation to be on the market. Indeed, Sun Microsystems is among the leaders and early movers of the RFID software field, which can be of great importance when talking about such a young technology.

The Sun Java System RFID Software is programmed in Java on top of JINI (see [17]) and RIO (see [31]). Based on the notion of "service", these two technologies are well suited to network RFID readers without (much) human involvement and in a very flexible manner. The interested

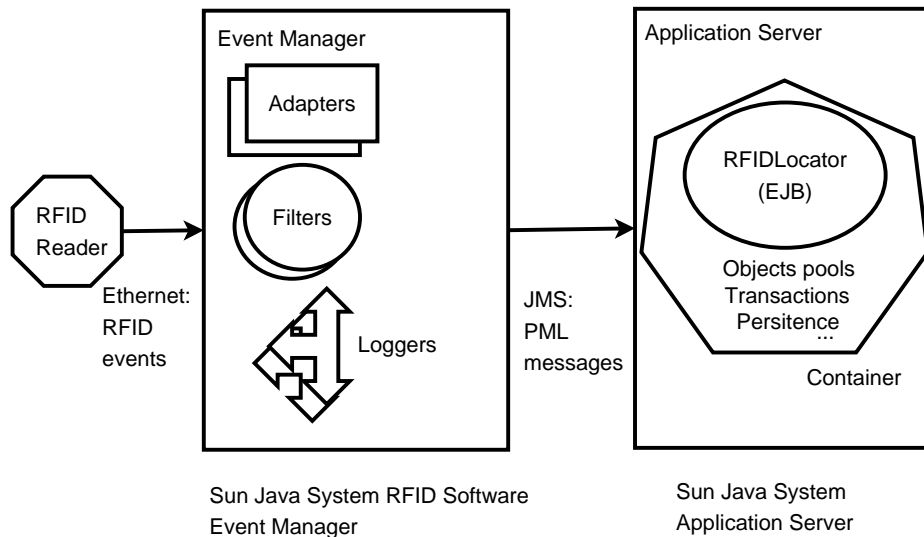


Figure 6.1.: The Event Manager and the Application Server

reader is invited to read [Gor04] for an introduction to the JINI technologies.

Finally, it is worth noting that the power of the Sun Java System RFID Software also resides in the fact that new Adapters, Filters and Loggers (see Section 3.4) can be programmed in Java and integrated easily with the existing RFID software. Additionally Figure 6.1 emphasizes the respective role of the Event Manager and the Application Server.

### 6.1.2. Enterprise JavaBeans

Nowadays many technologies propose to solve the problem of distributed computing. However, for enterprise applications, where robust and reliable software is a need, two technologies are leading the field. The first is the .NET Framework (see [6]) of Microsoft Corp. and the second is known as J2EE (see [14]). The war against the detractors and admirers of both technologies is certainly not over. However, in a concern of choosing an “open” standard, the J2EE and its “EJBs” (or Enterprise JavaBeans) were finally chosen.

This leaves a central question open: why use EJBs and not POJOs (Plain Old Java Objects)? Many elements form the answer:

First of all, the most convenient way to integrate the RFID Event Manager with external systems is to use JMS messages. Using such messages enables asynchronous processing of the events. In the case of an application like the RFIDLocator where the logged events are independent of the users, asynchronous processing seems to be the best answer. It is the best in terms of performances as well as in terms of simplicity.

Yet the integration of JMS messages in a distributed application is quite straightforward using Message Driven Beans components (see Section 5.3). This is the first argument towards the use of EJB.

Additionally, an application like the RFIDLocator might be subject to a great amount of concurrent calls. Think of such a tracking software installed in a big company or administration dealing with a huge number of documents. This could lead to several thousands of concurrent EPC tags observation and thus, thousands of concurrent processing requests. Programming the application in order for it to handle this fact is a solution, leaving the EJB container dealing with the load balancing and the scalability is certainly a better idea. Thus, this fact is also an argument for the use of the EJB standard.

Many other minor points are also counting towards the use of EJBs but all the arguments con-

verge to: “why reinventing the wheel?”. Enterprise JavaBeans enable a distributed application to be written without too many concerns about it being distributed. This fact leaves more time to the programmer to concentrate its work on the business logic, i.e. on the “components” rather than on the “artifacts”.

Eventually, it is worth noting that the version 2.0 (J2EE version 1.4) of the EJB standard was used for the RFIDLocator. The reason for using it and not the current 2.1 version is exposed in the next section.

Understanding the source code of this work suppose a basic knowledge of the basis of Enterprise JavaBeans. The book [MH04] provides the necessary knowledge to develop Enterprise Java Components (aka Enterprise JavaBeans). However, the article [Ort04] provides already a sufficient summary of the subject.

### 6.1.3. XDoclet

One of the common argument the detractors of J2EE (and its underlying EJB technology) use is the heaviness of the technology for the programmer. This argument is certainly not void and is somehow quite paradoxical. Indeed, where the programmer should be freed of programming the artifacts around its application, leaving her time to concentrate on the business logic, she wastes hours writing interfaces and deployment descriptors. However, one should note that writing the overhead imposed by the EJB standards is much simpler and much more elegant than writing load-balancers or transaction system for every new distributed application. Still finding a solution to minimize the “EJB overhead” seems to be a fair quest.

XDoclet provides a partial solution. Using this technology most of the interfaces and descriptors writing jobs can be automated through the use of metadata (or Doclets) embedded in the code. For instance Figure 6.2 exposes various XDoclet instructions. Consider the `@ejb.interface-method` instruction. It tells the XDoclet processor that the description of the Java method that will follow (i.e. `public abstract String getType();`) should be included in the auto generated interfaces. Where using XDoclet instructions drastically reduces the actual overhead it also adds some constraints particularly regarding the Java inheritance. However, for most enterprise applications the substantial gain overcomes the added constraints.

XDoclet-1.2.2 was used for generating the many interfaces and deployment descriptors of the RFIDLocator. It is worth noting that XDoclet is an open-source software that can be downloaded from [42].

Eventually, one should take into account that XDoclet is not the only possibility of reducing the “EJB Overhead”. For instance the version 4.1 of the NetBeans IDE automates the generation of interfaces and deployment descriptors without even a need for Doclet comments, reducing the overhead almost to zero.

This latter solution was not chosen mainly for pedagogical reasons. Indeed, with XDoclet one still realizes what is done “behind the scene” which is not really the case when using a totally automated process as provided by NetBeans 4.1<sup>1</sup>.

### 6.1.4. Application Server Software

The Application Server is a software on which a J2EE application can be installed (or deployed to be more precise). Because of the EJB specification, the choice of an Application Server should not be an irreversible decision. Indeed, any Application Server that implements the EJB 2.0 specification would theoretically be able to run the RFIDLocator without any changes (or after some minor changes in reality). However, the RFIDLocator is best designed for the Sun Java

---

<sup>1</sup>NetBeans IDE can be downloaded for free from [23]

---

```
/**
 * Gets the Type of Action (IN/OUT).
 *
 * @ejb.interface-method
 * @ejb.persistent-field
 * @ejb.persistence
 *   column-name="TYPE"
 */
public abstract String getType();
```

---

Figure 6.2.: A code extract containing XDoclet instructions

System Application Server (Platform Edition 8.1) coupled with a PointBase 4.8 RE RDBMS. Both are available for download from [14]. Among the other Application Server that should be able to run the RFIDLocator let us cite:

- Jonas (free of charge, downloads from [18]).
- JBoss Application Server (free of charge, available from [16]).
- IBM WebSphere (commercial application, trial downloads from [41]).
- Apache Geronimo Application Server (free of charge, downloads from [9]).

Similarly, about every RDBMS that can be accessed from a J2EE architecture is potentially able to host the database of the RFIDLocator. Among the the well-known databases that are well coupled with J2EE let us cite:

- MySQL (the Community License is free of charge, downloads available from [22]).
- Oracle Database (commercial application, trial and developer downloads available from [24]).
- PostgreSQL (open source database, free of charge, downloads from [27]).

## 6.2. Hardware Settings

The main goal of this project is definitely to test the RFID technology. Thus, working only with simulators would not have been really satisfactory. This is the main reason for installing a sample of the hardware enrolled in commercial applications to test the RFIDLocator. The purpose of this section is to expose the hardware settings deployed around the application. First by describing the components then by summarizing their relative roles in the overall process (see Section 6.3).

### 6.2.1. Readers and Connectors

#### Medio L100

The reader chosen for this work is a Medio L100 from Tagsys (see [37]), a French company expert in Auto-Id hardware. It provides two antennae with a range of about 20 cm each. It also encloses an operating system (GemCoreOS) which is in charge of a first filtering of the RFID events.

As shown on Figure 6.3, the Medio L100 is shipped with two output connectors. The first, an LPT (aka Parallel) port is provided for monitoring events before the built-in operating system gets a chance to filter them. The second, an RS-232 (aka Serial) port enables the retrieving of events after these went through the GEMCore-OS and its filtering capabilities.

This reader seems to be a good choice for testing applications not involving movements. However, after days of concrete tests we realized that it was not really the appropriate reader for the RFIDLocator. Indeed, the operating distances of the Medio L100 are quite disappointing when the tags are moving. As a consequence when testing the reading of tags in motion these should be placed really close to the antenna (that is at a maximum distance of 3 cm). Thus, for the purpose of the RFIDLocator, the Medio L100 is certainly not the most adequate choice but is still sufficient.

The technical documentation of both the Medio L100 and its built-in OS can be found on the CD-ROM of this application by following the path: `documentation/Tagsys`.

#### Moxa NPort Express

As we want to propagate the events reported by the reader to an Ethernet network (over TCP/IP), none of the ports provided by the Medio L100 is satisfactory. This is the main reason for setting a Moxa (see [21]) converter.

Such a piece of hardware acts as a (small) server taking data sent over an RS-232 port as input and encapsulating it into a TCP/IP packet. The whole is then redirected to an RJ-45 connector where it can be further propagated through the Ethernet protocol.

Plugged to the reader's Serial port on one side and to the University's LAN on the other, it makes the RFID events available on the network.

This is exactly what Event Manager server needs. This central component of the setting is the subject of the next section.

The technical documentation of the Moxa DE-311 can be found on the CD-ROM of this application by following the path: `documentation/Moxa`. A view of the actual Moxa used in this application is provided on Figure 6.4.

### 6.2.2. The Server of the Event Manager

According to the documentation (see [Sun04c]) provided by Sun, the Event Manager (see Subsection 3.4.1) can be installed on a Linux Red Hat or Solaris 9 powered machine.

To make the demonstration more realistic the Sun Java System RFID Event Manager is installed on a dedicated Ultra Enterprise 450 server from Sun running Solaris 9. The Server is



Figure 6.3.: Front view of the Medio L100



Figure 6.4.: The Moxa DE-311 Serial to TCP/IP converter

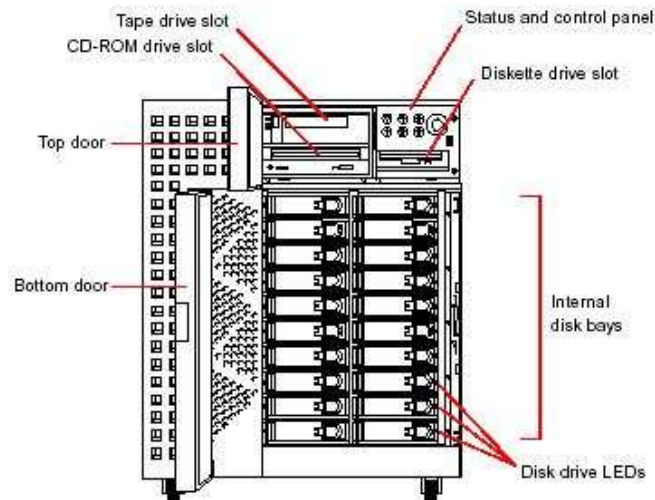


Figure 6.5.: Schematic view of the server (from [Sun97])

registered on the University network which enables it to catch the TCP/IP packets propagated by the Moxa and containing the EPC events.

This is the right place to thank Sun Microsystems Switzerland (see [35]) indeed, they offered us the Ultra Enterprise Server and all the required software licenses required by this machine.

Although the 4 motherboards and 20 hard-drive slots of this Ultra SPARC (II) machine are a bit of an overhead for the demonstration, it makes the whole closer to the kind of hardware enrolled in a real application. Indeed, the EM, central component of the EPC Network, might have to handle millions of events per second.

Eventually is worth noting that installing the Event Manager on a dedicated machine is not mandatory.

Figure 6.6 provides a picture of the actual machine used for the application whereas Figure 6.5 offers a schematic view of the elements. The Owner's Guide of the server can be found on the CD-ROM of the RFIDLocator by following the path: `documentation/UltraEnterprise`.

### 6.2.3. The Machine of the Application Server

Just like the Event Manager does not need to be installed on a dedicated machine, the Application Server holding the final application has no need to be setup on a separated machine. However, to be closer to a reality, the RFIDLocator is installed on a second dedicated machine. This computer is nothing but a PC (Personal Computer) running Linux Fedora Core 2 and the Sun Java System Application Server (Platform Edition 8.1). Figure 6.7 provides a picture of the actual computer (an IBM ThinkPad R40) used to act as the Application Server.<sup>2</sup>

### 6.2.4. RFID Labels

Last but not least, the RFIDLocator uses RFID tags embedded in thin stickers. Such tags are often known as "RFID Labels". The labels chosen for the application are EPC compliant and made together by Philips and Rafsec. Of about  $45\text{cm}^2$  these transponders are mainly intended for testing purposes. It is worth noting that although the RFIDLocator only uses tag reading, the transponders used here can be both read and written. A picture of the labels used by the RFIDLocator is provided on Figure 6.8.

<sup>2</sup>This latter machine was also used as a terminal for the Ultra Enterprise 450.



Figure 6.6.: The actual Ultra Enterprise 450 hosting the Event Manager



Figure 6.7.: The computer hosting the RFIDLocator

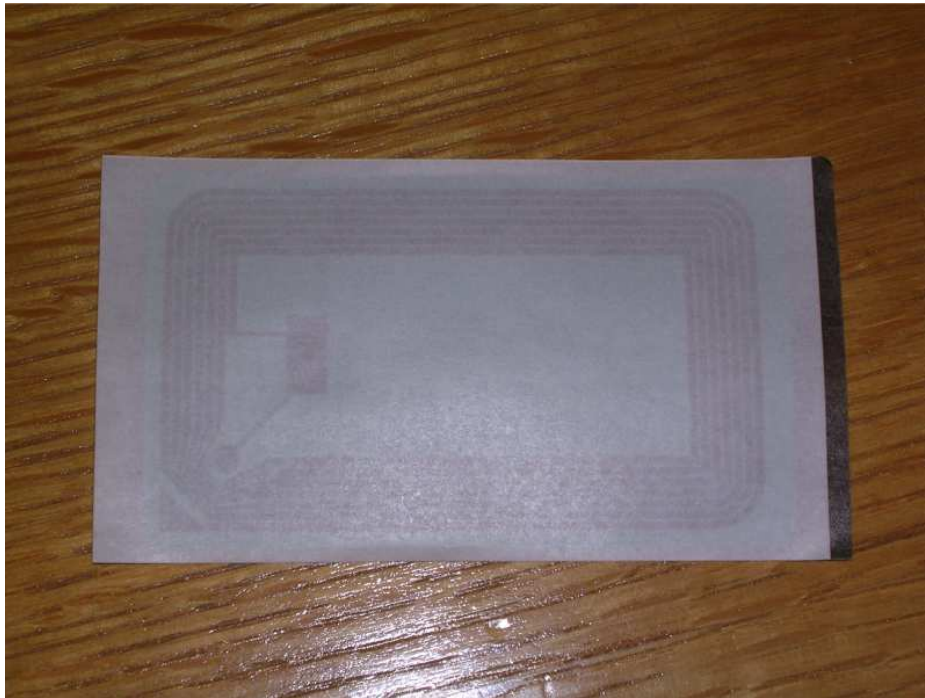


Figure 6.8.: Back view of a Philips Rafsec tag

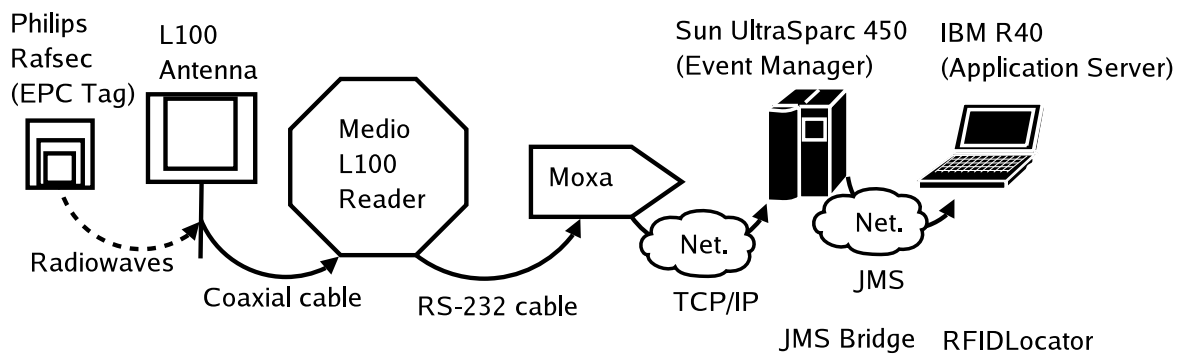


Figure 6.9.: The hardware devices of the demonstration

### 6.3. Summary

To help the reader to understand the settings, Figure 6.9 provides a summary of the most important hardware devices used for the the RFIDLocator. The reporting of an EPC event to the RFIDLocator goes through the following steps:

1. The antenna downloads the transponder's content (i.e. an EPC) as soon as the tag (Philips Rafsec) enters the electromagnetic field.
2. The antenna communicates the EPC to the reader through a coaxial cable.
3. The built-in OS of the reader (Medio L100) filters the events sent by the 1..n antennae and reports the events considered as being valid. A valid event at this level is one that can be distinguished from the noise captured by the antennae.
4. The valid events are reported by the reader on its serial port.

5. The Serial to TCP/IP converter (Moxa DE-311) connected to the reader catches the events and encapsulates them into TCP/IP packets.
6. The packets are propagated on the LAN<sup>3</sup> through Ethernet.
7. The server on which the Event Manager is installed (Sun Ultra Enterprise 450) receives the packets. This latter will then:
  - a) Use the suitable Adapter (Lx00Reader in our case).
  - b) Send the event through a Filter (RfidSmoother) which does about the same filtering as the reader's built-in OS.
  - c) Finally send the filtered events to the EpcGuiLogger (used for debugging purposes) and to the JMSLogger.

It is worth noting that the steps at the Event Manager level are described in more details in Subsection 3.4.1.

8. Installed on the Sun Ultra Enterprise 450, the JMSLogger is then going to:
  - a) Convert the EPC events (or Observations) into PML strings.
  - b) Encapsulate the PML into JMS packets, sending them to a JMS Queue (RFIDLocatorQueue). This latter Queue is a remote object as it is located on the machine holding the Application Server.
9. Eventually, the RFIDLocator installed on the machine of the Application Server (IBM ThinkPad R40) receives the PML and processes it. This latter processing is the matter of Section 4.

---

<sup>3</sup>On the University Network (unifrlan) to be more precise.

# 7

## RFIDLocator: Use and Installation

---

<b>7.1. User Manual</b> . . . . .	<b>75</b>
7.1.1. The Graphical User Interface . . . . .	75
7.1.2. Access to the Main Functionalities . . . . .	77
7.1.3. Test Users . . . . .	79
<b>7.2. Installation Guide</b> . . . . .	<b>82</b>
7.2.1. Installing the Event Manager . . . . .	82
7.2.2. Installing the RFID Reader . . . . .	83
7.2.3. Installing the Application Server and the Database . . . . .	83
7.2.4. JMS Settings . . . . .	87
7.2.5. Deploying the Application . . . . .	89
7.2.6. Starting the RFIDLocator . . . . .	90
<b>7.3. Extending the RFIDLocator</b> . . . . .	<b>91</b>
7.3.1. Required Software . . . . .	91
7.3.2. Compiling the project . . . . .	91

---

The aim of this chapter is to provide a basic user manual as well as a guide for the administrator of the application. Additionally it also contains some hints for the programmer interested to extend the RFIDLocator. Thus, it is divided into three sections. The first, to the attention of the reader, offers a description of the main use cases. The second, intended for the administrator provides an installation guide. Finally, the third contains an overview of the tasks required to extend and recompile the application.

### 7.1. User Manual

#### 7.1.1. The Graphical User Interface

The GUI (Graphical User Interface) of the RFIDLocator is a web based thin-client. To start it, one only needs to have a browser installed. The GUI has been tested successfully with several common browsers<sup>1</sup>. Besides, it is known that some bugs may appear when using Netscape Navigator (5 or later). However, since these bugs only affect the presentation of the page and not the business logic it should not be a problem.

When the browser is started the GUI can be accessed by typing<sup>2</sup>:

---

<sup>1</sup>It has been tested with: Mozilla Firefox 1.0 (win/linux), Opera 6 (win), Mozilla 1.7.6 (linux), Konqueror 3.2.2 (linux) and Microsoft Internet Explorer 6 (win).

<sup>2</sup>localhost can be replaced by any valid IP address or domain name if the HTTP Listeners of the Application Server are on and the server is configured on a network.



Figure 7.1.: The thin-client GUI of the RFIDLocator

<http://localhost:8080/RFIDLocator/index.html>  
in the address bar.

The welcome page (`home.jsp`) of the application appears<sup>3</sup>. Figure 7.1 presents a screenshot of it. All the pages of the user interface are based on the same template. As shown one the figure, they are divided into 6 main zones:

1. This is the logo of the application. A single click on it brings the user back to the welcome page (`home.jsp`).
2. The second zone contains two elements:
  - A link to `logout` (i.e. end the session of the current user).
  - A link to the `about.jsp` page, which contains links to the API and to the documentation of the application.
3. This is the main menu of the application. It offers an access to the main functionalities of the RFIDLocator. These use cases are briefly described in Subsection 7.1.2.
4. Zone 4 is the so called “body” of the page. The content of the page is displayed in this zone.
5. Zone 5 contains links and logos of the third-parties that participated to this project.
6. Zone 6 offers the same elements as Zone 3. It provides a quick access to the menu when the pages grow bigger.

<sup>3</sup>When starting the application for the very first time the classes are going to be compiled. Thus, it may take a few seconds before you see the pages appear.

The screenshot shows the RFID Locator web application interface. At the top, there is a navigation bar with the RFID Locator logo, a welcome message, and links for 'logout' and 'API, Documentation, etc. > about...'. Below the navigation bar is a menu on the left side with the following items:

- Create a new user > [add user](#)
- Configure the environment > [configure readers](#)
- Find a registered object > [seek traceable object](#)
- Attach or detach an RFID tag to an object > [attach/detach tag](#)
- Simulate PML events > [PML Simulator](#)

The main content area is titled 'Add a new user to the RFIDLocator'. It contains a form titled 'User's information' with the following fields:

- Location Id: 1
- Firstname: Dominique
- Lastname: Guinard
- Email: test@test.com
- Username: misterdom
- Password: bludu
- Role: CEO
- Employee number: gui-13443-ru-ce-ch
- Administrator:
- Active:

A 'Submit' button is located below the form. At the bottom of the page, there are logos for Software Engineering Group, Sun microsystems, and the University of Fribourg Faculty of Science. A footer navigation bar contains the following links: home | add user | configure readers | seek traceable object | attach/detach tag | PML simulator | about...

Figure 7.2.: Creating a new user

### 7.1.2. Access to the Main Functionalities

The main functionalities were already described in Chapter 4. Thus, this section only provides some details about them. All the functionalities are accessible using the menu on Zone 3 (see Figure 7.1).

#### Create a new User

This link permits to create a new user of the RFIDLocator. All the fields are required. Once the form is filled and submitted the new user can login via the home page<sup>4</sup>. Figure 7.2 offers a screenshot of this use case.

#### Configure the Environment

This page is required to set the initial environment of the RFIDLocator. An XML description of what reader is located where has to be provided in the text field. The syntax of the XML that must be used is described in Section 5.7.

<sup>4</sup>It is worth noting that the current user has to logout before a new user can login from the same computer (browser).

Welcome to the RFID Locator front end ...

RFID locator

API, Documentation, etc.  
> about...

www.gmipsoft.com.com/rfid

**Menu**

- Create a new user  
> [add user](#)
- Configure the environment  
> [configure readers](#)
- Find a registered object  
> [seek traceable object](#)
- Attach or detach an RFID tag to an object  
> [attach/detach tag](#)
- Simulate PML events  
> [PML Simulator](#)

**Configure the environment (readers)**

XML string for the reader's configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<RFIDLocator xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamesp
  <Reader brand="TagSys" model="Medio L100">
    <LogicalAntenna fiability="80">
      <Solver>
        <JNDIENCNam>ejb/DifferentActions Concrete Solver</JNDIENCNar
        <MaxReadingTime>5000</MaxReadingTime>
      </Solver>
      <Location>
        <BusinessLocationNumber>RM-S-3.113</BusinessLocationNumber:
        <Description>The office of P.Fuhrer and G.Mostefaoui</Description>
      </Location>
      <PhysicalAntenna id="urn:epc:id:gid:25.1.10">
        <Action>IN</Action>
      </PhysicalAntenna>
      <PhysicalAntenna id="urn:epc:id:gid:25.1.11">
        <Action>IN</Action>
      </PhysicalAntenna>
      <PhysicalAntenna id="urn:epc:id:gid:25.1.12">
        <Action>OUT</Action>
      </PhysicalAntenna>
    </LogicalAntenna>
  </Reader>
</RFIDLocator>
```

Submit

Software Engineering Group Sun microsystems UNIVERSITAS FIBURGENSIS FACULTY OF SCIENCE

home | [add user](#) | [configure readers](#) | [seek traceable object](#) | [attach/detach tag](#) | [PML simulator](#) | [about...](#)

Figure 7.3.: Readers' configuration page

Figure 7.3 provides a screenshot of the readers' configuration page.

**Hint:** For a faster configuration, you can simply modify the example that is provided by default. Note that if the submitted XML is invalid, the application will return a JAXB Exception.

### Attach / Detach a Tag

To inform the RFIDLocator that it should track an object, one needs to attach an RFID Tag to the object both physically and virtually. This page offers to do the latter.

First the user enters the BusinessNumber. As described in Section 4.2 it is a unique string that the company uses to identify the object. Examples of such strings are: case\_JP\_guinard\_05 or portable\_computer\_0205.

Then, the user enters the unique number of the RFID tag. It corresponds to the unique number recorded on the tag that was physically pasted onto the object.

Finally the user can choose between Attach and Detach. The former binds the tag id to the BusinessNumber, creating a Traceable Object. The latter deletes a Traceable Object from the system. It is worth noting that only the BusinessNumber is mandatory when Detach is selected.

The screenshot displays the RFID Locator web application interface. At the top left is the 'RFID locator' logo. The main header area contains the text 'Welcome to the RFID Locator front end ...' and navigation links for 'logout' and 'API, Documentation, etc. > about...'. The URL 'www.gmipsoft.com/.com/rfid' is visible in the top right. A left-hand menu lists several options: 'Create a new user > add user', 'Configure the environment > configure readers', 'Find a registered object > seek traceable object', 'Attach or detach an RFID tag to an object > attach/detach tag', and 'Simulate PML events > PML Simulator'. The main content area is titled 'Attach / Detach a Tag to a Business Object' and features a form for 'Business Object and RFID Tag Information'. The form includes a note: '\* Please note that all the fields are required.' and the following fields: 'BusinessNumber:' with the value 'espritDesign\_seminar\_021', 'EPC URI (or tag id):' with the value 'urn:epc:id:gid:19.90.113', and 'SecurityLevel:' with a dropdown menu set to 'confidential'. Below these fields are radio buttons for 'Detach' and 'Attach', with 'Attach' selected. A 'Submit' button is located at the bottom of the form. The footer of the page contains logos for 'Software Engineering Group', 'Sun microsystems', and 'UNIVERSITÄT FRIBURGENSIS FACULTY OF SCIENCE', along with a navigation bar and copyright information: 'Copyright 2005: Dominique Guinard: dominique.guinard@unifr.ch'.

Figure 7.4.: Attach / Detach a Tag

Once a tag is attached to an object it becomes a Traceable Object and thus, can be traced by the RFIDLocator.

Figure 7.4 offers a view of the Attach / Detach page.

### Find a Registered Object

This permits the user to seek a Traceable Object by providing its BusinessNumber. The system will return all the observations the RFID readers made from this particular object. Figure 7.5 presents the results of a typical seeking query.

### Simulates PML Events

This page offers to simulate the events reported by an RFID reader. It permits to test the application without having an actual physical reader. The user is prompted for a PML string that describes an observation. For more information on the syntax of the PML please refer to Section 3.3.

A view of the PML Simulator is provided on Figure 7.6.

#### 7.1.3. Test Users

A set of test users is provided with the application. Table 7.1 summarizes the passwords and usernames of the test set.

- > [configure readers](#)
- Find a registered object
- > [seek traceable object](#)
- Attach or detach an RFID tag to an object
- > [attach/detach tag](#)
- Simulate PML events
- > [PML Simulator](#)

Business number of the Traceable Object :

• Please enter the Business number of the Traceable Object you wish to seek for (e.g. madwWorlds\_Thesis):

Submit

Information about the Traceable Object: madwWorlds\_Thesis

**Attaching Date:**  
Sat Dec 04 00:00:00 CET 2004

**Business Number:**  
madwWorlds\_Thesis

**Electronic Product Code (EPC) or unique identifier:**  
urn:epc:id:gid:80.80.1

**Internal id (RFIDLocator wide) of this Object:**  
1

**Security level:**  
low

**Application's user identifier of the registrator:**  
10

Observations recoded for madwWorlds\_Thesis

Timestamp	Business location number	Description of the location	Action
Wed Jun 08 00:00:00 CEST 2005	RM-S-3.114	Office of Pr.J.Pasquier	Going <b>IN</b> RM-S-3.114
Thu Jun 09 00:00:00 CEST 2005	RM-S-3.114	Office of Pr.J.Pasquier	Going <b>OUT</b> RM-S-3.114
Wed Sep 07 00:00:00 CEST 2005	RM-S-3.114	Office of Pr.J.Pasquier	Going <b>IN</b> RM-S-3.114
Sun Dec 04 00:00:00 CET 2005	RM-S-3.114	Office of Pr.J.Pasquier	Going <b>OUT</b> RM-S-3.114





Figure 7.5.: Seeking a Traceable Object

Username	Password	Admin
test	test	yes
guinaldo	test0	yes
mireilleb	test1	no
jpg	test2	yes

Table 7.1.: Test users

Welcome to the RFID Locator front end ...

[logout](#) [API, Documentation, etc. > about...](#)

www.gmipsoft.com .com/rfid

### Menu

- Create a new user
  - [add user](#)
- Configure the environment
  - [configure readers](#)
- Find a registered object
  - [seek traceable object](#)
- Attach or detach an RFID tag to an object
  - [attach/detach tag](#)
- Simulate PML events
  - [PML Simulator](#)

### Simulate an RFID Reader: PML Simulator

PML Core string of the event to simulate:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Sensor xmlns="urn:autoid:specification:interchange:PMLCore:xml:schema:1">
  <ns1:ID xmlns:ns1="urn:autoid:specification:universal:Identifier:xml:schema:1">
    <!-- The urn below is the EPC of the Physical Antenna you want to simulate -->
    urn:epc:id:gid:1.1.1
  </ns1:ID>
  <Observation>
    <ns2:ID xmlns:ns2="urn:autoid:specification:universal:Identifier:xml:schema:1">
      5
    </ns2:ID>
    <!-- This specifies at what time the event occurred -->
    <DateTime>2005-04-14T13:13:00.812+02:00</DateTime>
    <Tag>
      <ns3:ID xmlns:ns3="urn:autoid:specification:universal:Identifier:xml:schema:1">
        <!-- This is the EPC of the tag that was seen -->
        urn:epc:id:gid:0.0.1
      </ns3:ID>
    </Tag>
  </Observation>
</Sensor>
```

home | add user | configure readers | seek traceable object | attach/detach tag | PML simulator | about...

Figure 7.6.: The PML Simulator

## 7.2. Installation Guide

Because of its distributed nature and due to required infrastructure around the core of the RFIDLocator, installing it is not as straightforward as setting up a fat client. This section, intended for the administrator of the application, offers a global view of the steps required to install the RFIDLocator and its infrastructure. Besides, a consequent part of the time spent on this Bachelor Thesis was related to the installation and integration of all the components. Thus, this section is also a summary of the work that was done before the actual programming of the application could start.

This guide supposes Ant installed on all the machines involved in the settings. Apache Ant is a Java-based build tool. It is similar to the well known Make tool but makes use of XML files and offers far more functionalities<sup>5</sup>.

### 7.2.1. Installing the Event Manager

As said before, this project uses the Event Manager of the Sun Java System RFID Software v.1.0. It is worth noting that just like most (if not all...) of the RFID middleware implementation, the Sun Java System RFID Software is not a free software. However, an evaluation version of the application can be ordered from [34]. The Event Manager from Sun requires either a Solaris (9 or later) platform or a Linux Redhat Enterprise machine. Even if not supported by Sun Microsystems, the installation of the EM on another Linux machine requires at worst some changes in the startup scripts.

The RFIDLocator was installed on a Solaris 9 Ultra-SPARC Server (see Section 6.2.2). Thus, it is this installation that is summarized below:

1. Copy the RFID\_SystemSoftware.SPARC.zip file to your Solaris (SPARC) machine.  
A copy of this file can be found on the CD-ROM (see Appendix C) by following the path: software/Sun.
2. The archive contains two components: the Event Manager and the Information Server. Install both as described in [Sun04c]. This document can be found on the CD-ROM at: documentation/Sun/SUN\_installation\_guide\_rfid\_soft\_1.pdf.  
**Hint:** The startup of the EM may cause troubles if it was not installed in the default directory (i.e. /opt/SUNWrfid). Thus, try to avoid installing it to another directory.
3. Once installed the Event Manager can be started using the startall script located in /opt/SUNWrfid/bin/<sup>6</sup>. Note that you need to be a superuser to start the EM.
4. At this step it is a good idea to test the installation. The PMLReader can be useful for this task. It is a software program that generates emulated RFID events. By default the EM is setup to listen to the PMLReader so all you have to do is starting it by typing the command PMLReader in /opt/SUNWrfid/bin/.  
More information about this emulator can be found in [Sun04c].  
**Hint:** If you are commanding the machine on which the EM was installed remotely, you will need to “export” the X11 display to your local machine in order for the PMLReader to work with the EPCGuiLogger (see [Sun04a]). This can be done by typing:  
xhost +  
on the commanding computer (client), and:  
export DISPLAY=IP\_ADDRESS\_OF\_THE\_EM:0.0  
on the machine holding the Event Manager (server).  
Where IP\_ADDRESS\_OF\_THE\_EM has to be replaced by the IP address of the commanding computer (client).

---

<sup>5</sup>This free software program can be downloaded from [1].

<sup>6</sup>To **stop** the Event Manager type **stopall** in the same directory.

Note that even if the Information Server is not required by the RFIDLocator, it is a good idea to install it. Indeed, it will install an implementation of JMS which is needed by the application. Of course, you may also use an existing implementation or install JMS separately. Eventually it is worth noting that the RFIDLocator works virtually with every Event Manager. The only requirement is for the implementation of the EM to be able to send the RFID events in the form of PML messages to a JMS Queue.

### 7.2.2. Installing the RFID Reader

Now that the Event Manager is set up we can install an RFID reader. The concrete tests were made using a Medio L100 from Tagsys but any EPC compliant reader can work with the RFIDLocator. As mentioned in Section 3.4, to add a new Reader to an EPC compliant Event Manager one should only need to install a new adapter.

However, as the Medio L100 does not offer to propagate the RFID events on a network, a Serial to Ethernet converter also needs to be installed (see Subsection 6.2.1). The settings of both these components are described in a document written by Sun Microsystems: [Sun04d]. A copy of this document is located on the CD-ROM at: `documentation/Sun/Tagsys_Adapter.pdf`. Two jar files are required: `lx00.jar` and `com.tagsys.jar`. A copy of both files can be found on the CD-ROM at: `software/Sun`.

As explained in the documentation, two files need to be configured:

- `RfidConfig.xml`: contains the configuration of the Adapters, Filters and Loggers of the Event Manager (see Section 3.4).
- `Rfid.xml`: contains the technical configuration of the Event Manager as well as the classes and jars to load at startup.

The `Rfid.xml` file can be edited as specified in [Sun04d]. For the `RfidConfig.xml` file the RFIDLocator needs special settings. The XML fragments that should be copied to the configuration file are provided in Figure 7.7 and 7.8. Additionally, a copy of both files (filled with all the configuration) is located on the CD-ROM at: `config/Event_Manager/`.

**Important notes:** To the contrary of what is explained in the documentation of the adapter (version 1), the EPC URI returned by the reader or the antennae can not be changed. Indeed, the adapter will always return `urn:epc:id:gid:1.1.1` and `urn:epc:id:gid:1.1.2` for the events captured by the Physical Antenna 1 respectively the Physical Antenna 2<sup>7</sup>.

Section 3.4 of the adapter's documentation (see [Sun04d]) tells you to modify the two configuration files but the given path is incorrect if you are using the version 1 of the RFID software. The valid path to find the files is: `/etc/opt/SUNWrfid/`.

### 7.2.3. Installing the Application Server and the Database

With the Event Manager installed we now need to setup an Application Server to run the RFIDLocator. As said before we used the Sun Java System Application Server (Platform Edition 8.1) but again, every J2EE compliant server should be able to run the application (with some minor settings and changes).

The Application Server from Sun can be downloaded for free from [33]. A copy of the linux version actually used for the project is provided on the CD-ROM of the project under: `software/Sun/j2eesdk-1.4.2005Q1-linux.bin`. The user-friendly install tool is started by running the `j2eesdk-1.4.2005Q1-linux.bin` binary.

Once installed the Sun Java System Application Server (Platform Edition 8.1) can be started

---

<sup>7</sup>This bug is solved in a new version of the Tagsys adapter. However, since this latter works only with the Sun Java System RFID Software version 2 it was not used for the project.

---

```

<!-- Configuration of the Medio L100 for the RFIDLocator -->
<ems:adapter>
<ems:name>Lx00Reader</ems:name>
  <ems:classname>com.sun.autoid.adapter.tagsys.Lx00ReaderAdapter
</ems:classname>
  <ems:properties>
    <ems:property>LogLevel</ems:property>
    <ems:value>FINEST</ems:value>
  </ems:properties>
  <ems:properties>
    <ems:property>hostname</ems:property>
    <ems:value>134.21.9.232</ems:value>
  </ems:properties>
  <ems:properties>
    <ems:property>port</ems:property>
    <ems:value>4001</ems:value>
  </ems:properties>
  <ems:properties>
    <ems:property>readerepc</ems:property>
    <ems:value>urn:epc:tag:gid-96:1.1.1</ems:value>
  </ems:properties>
  <ems:properties>
    <ems:property>TagType</ems:property>
    <ems:value>EPC</ems:value>
  </ems:properties>
  <ems:properties>
    <ems:property>AntennaSequence</ems:property>
    <ems:value>ANT0,ANT1</ems:value>
  </ems:properties>
  <ems:properties>
    <ems:property>Power</ems:property>
    <ems:value>1000</ems:value>
  </ems:properties>
  <ems:properties>
    <ems:property>WaitBeforeScanning</ems:property>
    <ems:value>250</ems:value>
  </ems:properties>
  <ems:properties>
    <ems:property>autoread</ems:property>
    <ems:value>true</ems:value>
  </ems:properties>
  <ems:outputs>
    <ems:output>RfidSmoother</ems:output>
    <!-- comment the following line if you do not want to see the
output of the Medio L100 in the EPC GUI Logger -->
    <ems:output>EpcGuiLogger</ems:output>
    <ems:output>JMSLogger</ems:output>
  </ems:outputs>
</ems:adapter>

```

---

Figure 7.7.: Medio L100 configuration for the RFIDLocator

---

```
<!-- configuration of the JMS for the RFIDLocator -->
<ems:logger>
  <ems:name>JMSLogger</ems:name>
  <ems:classname>com.sun.autoid.logger.JMSLogger</ems:classname>
  <ems:properties>
    <ems:property>LogLevel</ems:property>
    <ems:value>CONFIG</ems:value>
  </ems:properties>
  <ems:properties>
    <ems:property>JndiContextFactory</ems:property>
    <ems:value>com.sun.jndi.fscontext.RefFSContextFactory
    </ems:value>
  </ems:properties>
  <ems:properties>
    <ems:property>JndiProviderURL</ems:property>
    <ems:value>file:///tmp</ems:value>
  </ems:properties>
  <ems:properties>
    <ems:property>ConnectionFactory</ems:property>
    <ems:value>MyQueueConnectionFactory</ems:value>
  </ems:properties>
  <ems:properties>
    <ems:property>QueueName</ems:property>
    <ems:value>MyQueue</ems:value>
  </ems:properties>
</ems:logger>
```

---

Figure 7.8.: JMS Logger configuration for the RFIDLocator

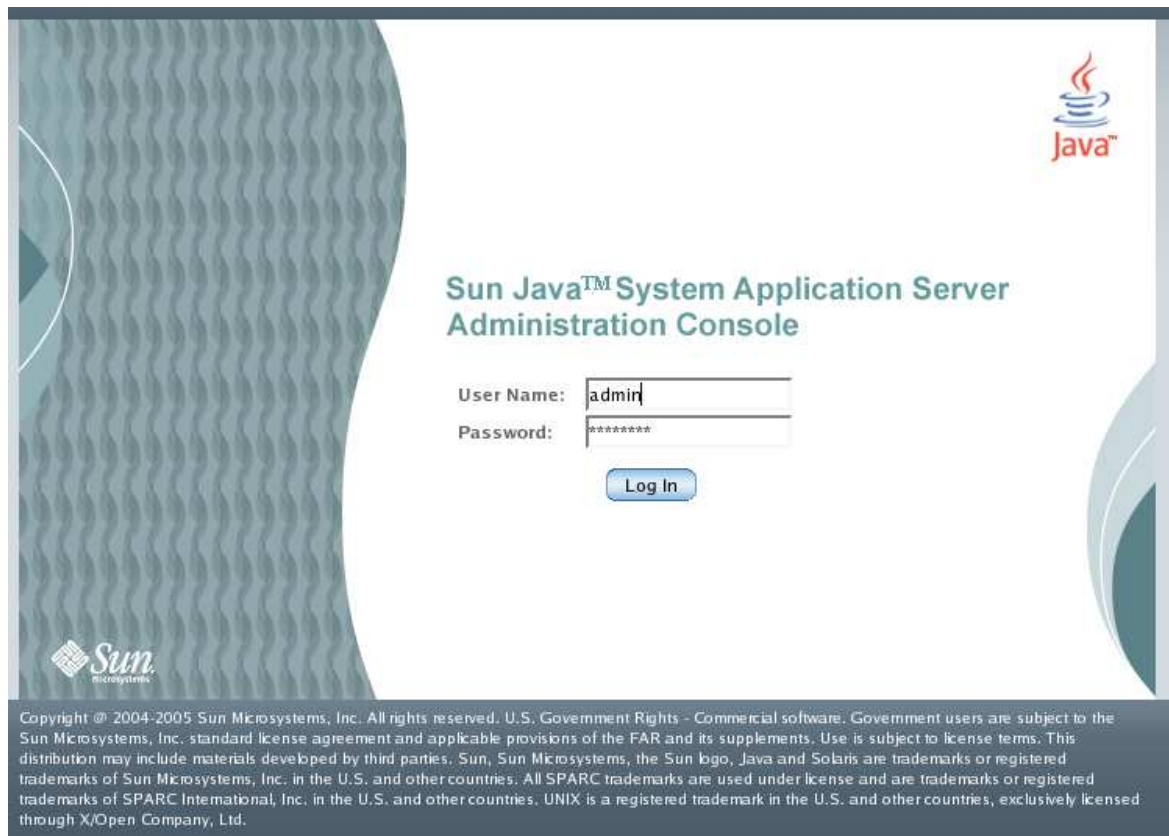


Figure 7.9.: Welcome screen of the Sun Java System Application Server

Field	Value
DatabaseName	jdbc:pointbase:server://localhost:9092/sample
Password	pbPublic
User	pbPublic

Table 7.2.: Physical Destinations required by the RFIDLocator

by typing<sup>8</sup>:

```
/AS_INSTALL_DIR/SUNWappserver/bin/asadmin start-domain -verbose=true domain1
```

To test the installation open a new browser window and type<sup>9</sup>:

```
http://localhost:4848
```

If the installation was a success, you should see the login page of the administration console within a few seconds. This page is shown on Figure 7.9.

The PointBase RDBMS (Relational DataBase Management System) is automatically installed with the Application Server. Two databases are also set at startup. The RFIDLocator uses one of them called sample. To check if this database is correctly configured in the Application Server go to the administration console (<http://localhost:4848>), login and browse to Application Server > Resources > JDBC > Connection Pools > PointBasePool. Under Properties enter the settings as specified on Table 7.2 (if not already set). Note that you may need to restart the Application Server after this step was performed.

<sup>8</sup>Replace AS\_INSTALL\_DIR by the path to your installation of the Application Server

<sup>9</sup>Provided a default installation was made.

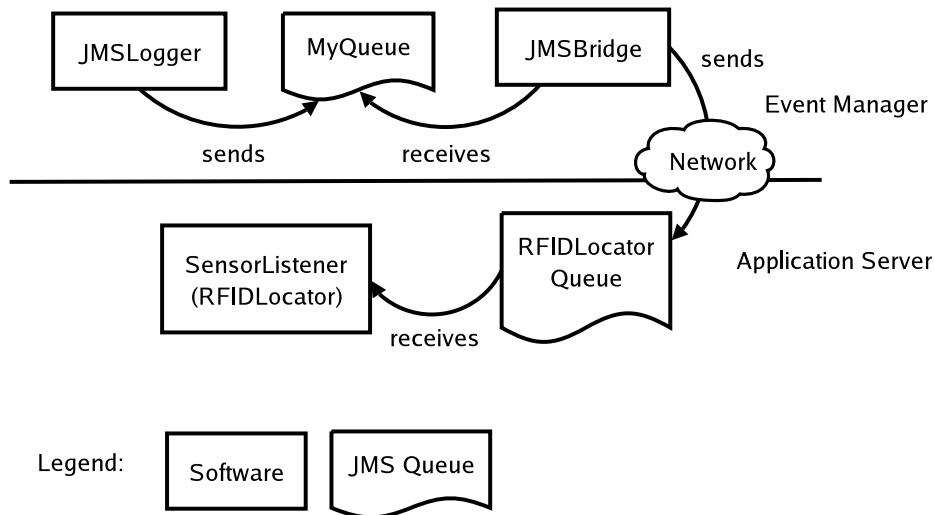


Figure 7.10.: The JMS queues of the RFIDLocator

### 7.2.4. JMS Settings

As mentioned in Chapter 4, the communication between the Event Manager and the RFIDLocator is using asynchronous JMS messages. Thus, the infrastructure of the RFIDLocator needs to include various JMS components. This Subsection summarizes the JMS elements to configure. Additionally, the interactions between these components is shown on Figure 7.10.

#### JMS Queue on the Event Manager

The RFIDLocator processes RFID events in the form of JMS PML messages. Thus, in order for the Event Manager to send the events to a JMS queue we first need to configure the JMS destinations. This procedure is described Chapter 6 of [Sun04a] which can also be found on the CD-ROM of the application at: [documentation/Sun/SUN\\_rfid\\_admin\\_guide\\_817-7896-10.pdf](documentation/Sun/SUN_rfid_admin_guide_817-7896-10.pdf).

The JMS components to configure are summarized on Table 7.3. As configuring the queues on the EM is very similar to the configuration on the AS, you may also use the procedure described below.

#### JMS Queue on the Application Server

A JMS queue also needs to be configured on the Application Server. We will describe the required steps briefly:

1. Start the Application Server (if not already started).
2. Open a new browser window and go to: <http://localhost:4848>. The systems prompts for username and password. Within a few seconds you should gain access to the administration console of the Sun Java System Application Server (Platform Edition 8.1).
3. Select Resources)JMSResources)ConnectionFactory and click on new.
4. For the JNDI Name enter: `jms/RFIDLocatorQueueConnectionFactory` and select `QueueConnectionFactory` for the type. You can now press “ok”. A screenshot of this step is provided on Figure 7.11.

This first step creates a new Connection Factory, an object used for clients willing to connect to a queue.

The next steps configure a new Queue:

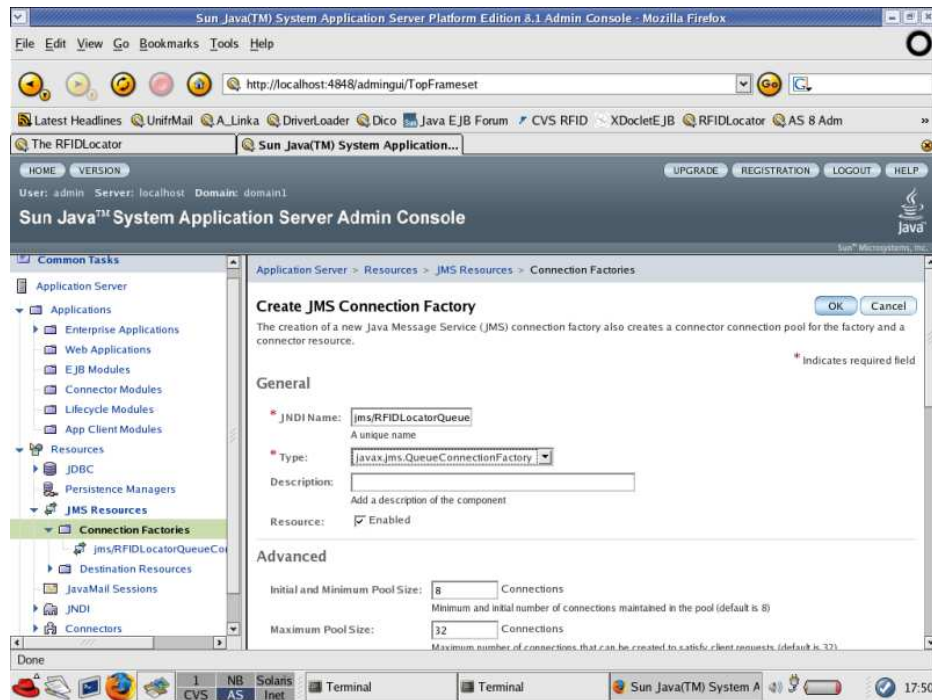


Figure 7.11.: Configuring a new Connection Factory

1. Select Resources)JMSResources)DestinationResources from the left menu.
2. Again, click on “new”.
3. Enter jms/RFIDLocatorQueue for the JNDI Name and select javax.jms.Queue for the type.
4. For the Value of the Name property enter RFIDLocatorPQ. This is the name of the actual Queue abstracted by the jms/RFIDLocatorQueue.

The last steps create the Physical Destination (i.e. the actual “queue” that will be maintained by the Sun Java System Message Queue brokers):

1. Select Configuration)JavaMessageService)PhysicalDestinations from the left menu.
2. A click on “new” prompts you for more information.
3. Enter RFIDLocatorPQ for the Physical Destination Name and select Queue for the type.

The JMS elements needed by the Application Server are now set. Tables 7.3 and 7.4 provide a summary of the JMS elements that must be configured on both the Event Manager and the Application Server in order to run the RFIDLocator with physical RFID readers.

It is worth noting that if you plan to use the PML Simulator only there is no need to configure the JMS objects on the EM. Indeed, the simulator sends the messages directly to the Queue of the AS.

## JMS Bridge

As the RFIDLocator does not need for the Application Server and the Event Manager to be located on the same physical machine (which is “a must” when deploying the application in a well-sized company...), a JMS Bridge between the queue of the AS and the queue of the EM was programmed. It is worth noting that such a bridge is theoretically not required. However since we were not able to make the JMS Logger of the Event Manager connect itself to a remote

JNDI Name	Type	EM or AS
jms/RFIDLocatorQueue	javax.jms.Queue	AS
jms/RFIDLocatorQueue ConnectionFactory	QueueConnectionFactory	AS
MyQueue	javax.jms.Queue	EM
MyQueueConnectionFactory	QueueConnectionFactory	EM

Table 7.3.: Queues and Connections Factories used by the RFIDLocator

Physical Name	Destination	Type	EM or AS
RFIDLocatorPQ		queue	AS
MyQueueDest		queue	EM

Table 7.4.: Physical Destinations required by the RFIDLocator

queue, the bridge was our only solution.

The JMS Bridge is a POJO that listens to a queue and transmit the incoming messages to another. To install it, simply copy the folder `software/RFIDLocator/JMSBridge` that can be found on the CD-ROM, to the machine on which the Event Manager was installed. Once copied, it is started by typing:

```
ant run
```

in the `JMSBridge` folder. The `JMSBridge` was programmed to run in an infinite loop. Thus, to stop it hit:

```
CTRL + C
```

in the terminal window.

### 7.2.5. Deploying the Application

The core of the `RFIDLocator` now needs to be deployed on the Application Server. This task needs only to be done once as the server will start the application automatically at each subsequent startup.

- First start the Application Server:  

```
/AS_INSTALL_DIR/SUNWappserver/bin/asadmin start-domain -verbose=true domain1
```
- and the PointBase RDBMS:  

```
/AS_INSTALL_DIR/SUNWappserver/SUNWappserver/pointbase/tools/serveroption/startserver.sh
```
- Then, copy the folder `software/RFIDLocator/RFIDLocatorEJB` (located on the CD-ROM ) to the machine on which the Application Server was installed.
- Browse (using the command line) to `RFIDLocatorEJB/` and edit the `build.xml` file (a simple text editor suffices). On the line:  

```
<property name="j2ee.dir" value="/home/misterdom/SUNWappserver">
```

 replace `/home/misterdom/SUNWappserver` by the path to the Sun Java System Application Server (Platform Edition 8.1) on your machine.
- Save the file and type:  

```
ant create-db-tables
```

 to create the database tables and test sets of the application

- Finally, type:  
ant deploy  
to deploy the RFIDLocator on the server.

After some compiling and checking the RFIDLocator should be deployed on the Application Server.

### 7.2.6. Starting the RFIDLocator

To start the RFIDLocator, the underlying Event Manager and Application Server follow the steps listed below:

#### 1) Starting Application Server and the Application

1. Start the Application Server:  
`/AS_INSTALL_DIR/SUNWappserver/bin/asadmin start-domain -verbose=true domain1`
2. Start the PointBase RDBMS:  
`/AS_INSTALL_DIR/SUNWappserver/SUNWappserver/pointbase/tools/serveroption/startserver.sh`
3. If not already done, deploy the RFIDLocator on the Application Server (see Subsection 7.2.5).

#### 2) Starting the Event Manager and the Bridge

1. Turn on the Moxa Converter previously plugged on the Medio L100 RFID reader.
2. Connect the antennae to the reader.  
**Warning:** Turning on the reader before connecting the antennae may cause damages to the Medio L100.
3. Turn on the Medio L100 Reader.
4. Start the Event Manager:  
`/opt/SUNWrfid/bin/startall`  
Note that it may take some minutes (usually up to two minutes) for the readers to be started. When the Medio L100 starts you should hear two short beeps<sup>10</sup>.
5. Start the JMSBridge by running  
ant run  
in the JMSBridge folder of the computer holding the Event Manager (see Subsection 7.2.4).

The infrastructure is now ready. The RFIDLocator can start to process RFID Events. To test the settings try to walk with some RFID tags in front of the antennae.

It is worth noting that the RFIDLocator can also be started without a running Event Manager. Indeed, since the RFIDLocator provides a PML Simulator (see Subsection 5.2.5) a physical reader is not mandatory to test the core of the RFIDLocator.

Remember that the main purpose of the Event Manager is to run the physical readers, as a consequence when using the PMLSimulator only, an Event Manager does not even need to be installed.

---

<sup>10</sup>It is worth noting that the Event Manager can also be turned on before the RFID Reader as long as the Moxa Converter is started.

## 7.3. Extending the RFIDLocator

This section does not provide a guide on how to extend the application. However it offers some keys required by the programmer willing to add functionalities to the RFIDLocator. The prerequisites of this section are a good knowledge of both Ant and Java (especially J2EE). The official manual of the former can be accessed on-line from [2].

### 7.3.1. Required Software

Besides the softwares listed in the Installation Guide, re-compiling the RFIDLocator requires some more application softwares:

- XDoclet has to be installed, it can be downloaded for free from [42]. The project was developed using the version 1.2.2.
- If you plan to change the Reader's Configuration Formalism (see Subsection 5.7), you will need to re-generate the XML Data Binding classes. The Java Web Services Developer Pack offers a good way (the JAXB compiler) of generating these classes. It can be downloaded for free from [15].

Besides, as the project was developed with the Netbeans IDE 4.0 (available for free from [23]), it might be a good idea to install this IDE as well. However, since the project is based on non-proprietary Ant build files the use of Netbeans is not mandatory.

### 7.3.2. Compiling the project

The application is composed of two separated projects:

- The core application (the RFIDLocator) whose files are located on the CD-ROM under: `software/RFIDLocator/RFIDLocatorEJB`.
- The JMSBridge whose files are located under: `software/RFIDLocator/JMSBridge`

Both projects have their own Ant `build.xml` file.

To recompile the JMSBridge use the `compile` target:

```
ant compile
```

To run it issue:

```
ant run
```

The RFIDLocator is a bigger application and thus many Ant targets are provided. Before using them you must change some settings in the `build.xml` file located at the root of the project. Browse to the following lines:

---

```
<!-- 2) To change for extending or recompiling the RFIDLocator -->
<property name="xdoclet.dir" value="/home/misterdom/xdoclet-1.2.2"/>
<property name="jwsdp.dir" value="/home/misterdom/jwsdp-1.5"/>
```

---

and change the values of both properties to point to the root folder where you installed XDoclet, respectively the Java Web Service Developer Pack<sup>11</sup>.

With this set, you may now use the various Ant targets. Among these let us cite the three most important:

---

<sup>11</sup>As said before only XDoclet is mandatory to recompile the project.

- 
- **ant compile:** compiles all the classes, generates the interfaces and the deployment descriptors. For this generation the target uses the XDoclet metadata embedded in the source code.
  - **ant dist:** generates the distributable files and the database.
  - **ant deploy:** copies the .ear distributable file of the project to the autodeploy directory of the Application Server where it is going to be deployed after a few seconds.

# 8

## Conclusion

### 8.1. Possible Extensions

The RFIDLocator is a working prototype of an Assets Tracking application. It is not purely a theoretical application as it reflects a need that many organizations or administrations formulated. However, to make it fully usable in a real environment some more work is required.

As described in Section 4.3, this version of the RFIDLocator implements the central algorithms, the kernel of the application and the core end-user uses cases. Indeed some quite useful (but not mandatory) functionalities were left for further extensions.

Among these, the use cases related to the management of the environment seem to be of particular interest. Indeed, the current form of the application enables the user to submit the environment in the form of an XML stream. However, once this is done, the user has no way of modifying the reader's configuration environment. Thus, providing the user with a GUI enabling her to modify the initially parsed configuration file seems to be a good idea. It would permit a lot more flexibility and would also extend the possible uses of the application.

Besides, the tasks related to the users management were implemented only partially. For instance, a finer granularity in the roles of the RFIDLocator users could be of great help in a real environment. The deprovisioning of Users might also be useful.

Eventually, the front end (Java Server Pages, or JSP) could be improved. Indeed, cleaner login processes and better user input checks might be achieved using the Intercepting Filter pattern (see [ACM03]). Besides, a greater separation of the presentation and the business logic in the JSP pages is a good idea. This can be accomplished using Java Beans and Taglibs or a presentation framework like Struts (see [12]).

### 8.2. Final Thoughts about the RFID Technologies

“I am sick and tired of being monitored on what I purchase, where I go, what I drive, how fast or slow it takes me to get there and data gathering agencies profiting from that makes it even worse. If they want data, have them send me a survey. I will answer what I want, but then let me INVOICE them. If anyone is going to make money on my information let it be me!”

- *Anonymous CASPIAN member, 2005*

“Radio frequency is another technology that supermarkets are already using in a number of places throughout the store. We now envision a day where consumers will walk into a store, select products whose packages are embedded with small radio frequency UPC codes, and exit the store without ever going through a checkout line or signing their name on a dotted line.”

- *Jacki Snyder, Manager of Electronic Payments for Supervalu (Supermarkets), Inc.*

Two quotes, two drastically different visions of our future with (or without ?) the RFID technology.

During my work on this project I had plenty of occasions to experience the fears of the common people regarding this new assets tracking technology. Many of my friends are evolving in a technological field. Surprisingly enough, even they seem to have quite a negative point of view about the RF Identification. As revealed in surveys made in Europe and in the USA only about half of the folk thinks the use of the RFID is a good thing (see [O'C05c]).

As a matter of fact the RFID technology has a fantastic potential: within the supply chain but also for other more creative applications. Still, the opinion of the people is crucial for a new technology. Many managers of the distribution market all over the world have the feeling that the RF technologies will be introduced with or without the approval of the customers. In my opinion this way of thinking is fairly dangerous. When talking about the RFID, the consumers have the mass media with them. These are the relays of the doubts and fears about the technology. Almost no article, no TV or radio show talking about the RF technologies forget to mention the privacy concerns of the consumers if it was to be introduced massively.

Thus, I think it is about the time for the RFID actors to realize that they must not only sell this technology to managers but also to the common people. They should emphasize the “bright side” of the idea. They ought to demonstrate the ability of networking all the objects surrounding us as a way of facilitating our lives<sup>1</sup> not only as a way of optimizing the supply chain and the distribution. Talking about this last point, they should also be fair and honest: in this field, it has been proven that in many cases the RFID permits a positive ROI (Return On Investment). Reporting a part of this benefit on the prices is both a way to be fair to the consumer and a good mean of showing the positive side of the technology.

Besides, there is a strong need for clear laws and recommendations about the tracking of goods, animals and even people (see [O'C05a]). This would restore confidence among the folk and permit the governments to crack down when needed. Finally, it would also oblige the distributors to be transparent, a really important point revealed by many surveys.

Because, yes, the RFID can be misused but which technology can not be?

---

<sup>1</sup>See [SFV04], [O'C05b], [ACM05] or ... the RFIDLocator for good examples of the RFID facilitating our lives.

# A

## Common Acronyms

Acronym	Definition
ACII	American Standard Code for Information Interchange
API	Application Programming Interface
API	Application Programming Interface
ASP	Active Server Page
AS	Application Server
AutoID	Automatic Identification
CASPIAN	Consumers Against Supermarket Privacy Invasion and Numbering
CTO	Composite Transfer Object
CPU	Central Processing Unit
DNS	Domain Name Service
EEPROM	Electrical Erasable Programmable Read Only Memory
EAN	European Article Number
EM	Event Manager
ENC	Environment Naming Context
EPC	Electronic Product Code
EPCG	EPC Global
EPCIS	Electronic Product Code Information System
ERP	Enterprise Resource Planning
GIAI	Global Individual Asset Identifier
GID	General IDentifier
GLN	Global Location Number
GPS	Global Positioning System
GRAI	Global Returnable Asset Identifier
GTIN	Global Trade Item Number
GUI	Graphical User Interface
HTML	HyperText Markup Language
IBM	International Business Machines
IS	Information Server
J2EE	Java 2 Platform, Enterprise Edition
J2SE	Java 2 Platform, Standard Edition
JAR	Java ARchive
JAXB	Java Architecture for XML Binding

Acronym	Definition
JDOM	Java Document Object Model
JINI	Pronounced like “genie”. It is a pseudo-acronym for: Jini Is Not Initials
JMS	Java Messaging Service
JNDI	Java Naming and Directory Interface
JSP	Java Server Pages
JWSDP	Java Web Services Developer Pack
LAN	Local Area Network
MDB	Message Driven Bean
NAPTR	Naming Authority PoinTeRs
OCR	Optical Character Recognition
ONS	Object Name Service
PC	Personal Computer
PHP	PHP Hypertext Preprocessor (formerly Personal Home Page)
PML	Physical Markup Language
POJO	Plain Old Java Object
RAM	Random Access Memory
RDBMS	Relational Data Base Management System
RFID	Radio Frequency IDentification
RF	Radio Frequency
RIED	Real-time In-memory Event Database
ROM	Remote Access Memory
RPC	Remote Procedure Calls
SQL	Structured Query Language
SSCC	Serial Shipping Container Code
SPARC	Scalable Performance ARChitecture
TMS	Task Management System
TO	Transfer Object
UCC	Uniform Code Council
URI	Uniform Resource Locator
WSDL	WebService Definition Language
WS	WebService
WWW	World Wide Web
XML	eXtended Markup Language
XSD	XML Schema Definition

# B

## XML Schema

### B.1. PML Schema

The PML Core XML Schema of this Appendix are the property of the EPCglobal. They are provided for informational purposes only. The original schema can be found at [7]

The first schema, PmlCore.xsd contains the syntactic definition of the root element of the PML language (i.e. the Sensor element). The second schema, Identifier.xsd provides a validation schema for unique identifiers to be used within the EPC Network.

#### B.1.1. PmlCore.xsd

---

```
<?xml version="1.0" encoding="UTF-8"?>
<schema targetNamespace="urn:autoid:specification:interchange:PMLCore:xml:schema:1"
xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:autoid="http://www.autoidcenter.org/2003/xml"
xmlns:pmlcore="urn:autoid:specification:interchange:PMLCore:xml:schema:1"
xmlns:pmluid="urn:autoid:specification:universal:Identifier:xml:schema:1"
elementFormDefault="qualified"
attributeFormDefault="unqualified" version="1.0">
<import namespace="urn:autoid:specification:universal:Identifier:xml:schema:1"
schemaLocation="../../Universal/Identifier.xsd"/>
<annotation>
<documentation>
<autoid:copyright>Copyright 2003 Auto-ID Center, All Rights Reserved.
</autoid:copyright>
<autoid:disclaimer>Auto-ID Center, its members, officers, directors, employees,
or agents shall not be liable for any injury, loss, damages, financial or
otherwise, arising from, related to, or caused by the use of this
document. The use of said document shall constitute your express consent to the
foregoing
exculpation.</autoid:disclaimer>
<autoid:program>Auto-ID version 1.0</autoid:program>
<autoid:purpose>PML Core Specification version 1.0</autoid:purpose>
</documentation>
</annotation>
<element name="Sensor" type="pmlcore:SensorType"/>
<complexType name="AnyXMLContentType">
```

```

<annotation>
<documentation>
<autoid:definition>The AnyXMLContentType provides localized openness
</autoid:definition>
</documentation>
</annotation>
<sequence>
<any namespace="##any" processContents="skip">
<annotation>
<documentation>
<autoid:definition>Any content</autoid:definition>
</documentation>
</annotation>
</any>
</sequence>
</complexType>
<complexType name="DataType">
<annotation>
<documentation>
<autoid:definition>The Data element holds text, binary or XML data.
</autoid:definition>
</documentation>
</annotation>
<choice>
<element name="Text" type="string">
<annotation>
<documentation>
<autoid:definition>Text value</autoid:definition>
</documentation>
</annotation>
</element>
<element name="Binary" type="hexBinary">
<annotation>
<documentation>
<autoid:definition>Binary value</autoid:definition>
</documentation>
</annotation>
</element>
<element name="XML" type="pmlcore:AnyXMLContentType">
<annotation>
<documentation>
<autoid:definition>The XML element holds any XML elements the instance author
would like to include. It is provided to enable localized openness and to allow
instance document authors to create
instance documents containing elements above and beyond what is specified by
the PML CORE
schema</autoid:definition>
</documentation>
</annotation>
</element>
</choice>

```

```
</complexType>
<complexType name="ObservationType">
  <annotation>
  <documentation>
  <autoid:definition>Information related to an observation/measurement by a sensor
  in the EPC Network. Observations represent measurements by the sensor.
  They associate the actual observed data with the
  sensor.</autoid:definition>
  </documentation>
  </annotation>
  <sequence>
  <element ref="pmluid:ID" minOccurs="0">
  <annotation>
  <documentation>
  <autoid:definition>The observation ID element is a number assigned to this
  specific
  observation.</autoid:definition>
  </documentation>
  </annotation>
  </element>
  <element name="DateTime" type="dateTime">
  <annotation>
  <documentation>
  <autoid:definition>The Observation DateTime element denotes the date and time
  stamp when the observation was made.</autoid:definition>
  </documentation>
  </annotation>
  </element>
  <element name="Command" type="string" minOccurs="0">
  <annotation>
  <documentation>
  <autoid:definition>The observation command element denotes the command was
  issued to the sensor to trigger the observation.</autoid:definition>
  </documentation>
  </annotation>
  </element>
  <element name="Tag" type="pmlcore:TagType" minOccurs="0" maxOccurs="unbounded">
  <annotation>
  <documentation>
  <autoid:definition>The Observation Tag element denotes tags observed by a sensor
  as part of the observation.</autoid:definition>
  </documentation>
  </annotation>
  </element>
  <element name="Data" type="pmlcore:DataType" minOccurs="0" maxOccurs="unbounded">
  <annotation>
  <documentation>
  <autoid:definition>The Observation Data element denotes any data captured by the
  sensors as part of the observation.</autoid:definition>
  </documentation>
  </annotation>
  </element>
  </sequence>
  </complexType>
```

```
</element>
</sequence>
</complexType>
<complexType name="SensorType">
  <annotation>
    <documentation>
      <autoid:definition>Information related to a sensor in the EPC Network. A sensor
      is any device that is capable of making measurements e.g. RFID readers,
      temperature sensors, humidity sensors.</autoid:definition>
    </documentation>
  </annotation>
  <sequence>
    <element ref="pmluid:ID">
      <annotation>
        <documentation>
          <autoid:definition>The Sensor ID element is the number assigned to this
          particular sensor in the EPC network. It is by default an EPC.
          If a different identification scheme is to be used, the identification
          scheme must be specified using the attributes of the identifier type.
        </autoid:definition>
        </documentation>
      </annotation>
    </element>
    <element name="Observation" type="pmlcore:ObservationType"
      maxOccurs="unbounded">
      <annotation>
        <documentation>
          <autoid:definition>The Sensor Observation element denotes
          observations/measurements
          made by this particular sensor.</autoid:definition>
        </documentation>
      </annotation>
    </element>
  </sequence>
</complexType>
<complexType name="TagType">
  <annotation>
    <documentation>
      <autoid:definition>Information related to a tag in the EPC Network.
      A tag is any electronic or nonelectronic
      device that carries at least an identifier.</autoid:definition>
    </documentation>
  </annotation>
  <sequence>
    <element ref="pmluid:ID">
      <annotation>
        <documentation>
          <autoid:definition>The Tag ID element is a unique number assigned to the
          tag.</autoid:definition>
        </documentation>
      </annotation>
    </element>
  </sequence>
</complexType>
```

```

</element>
<element name="Data" type="pmlcore:DataType" minOccurs="0">
<annotation>
<documentation>
<autoid:definition>The Tag Data element contains any data stored on the
tag.</autoid:definition>
</documentation>
</annotation>
</element>
<element ref="pmlcore:Sensor" minOccurs="0" maxOccurs="unbounded">
<annotation>
<documentation>
<autoid:definition>The Tag Sensor element denotes any sensor that is
mounted on the
tag</autoid:definition>
</documentation>
</annotation>
</element>
</sequence>
</complexType>
</schema>

```

---

### B.1.2. Identifier.xsd

---

```

<?xml version="1.0" encoding="UTF-8"?>
<schema
targetNamespace="urn:autoid:specification:universal:Identifier:xml:schema:1"
xmlns:pmluid="urn:autoid:specification:universal:Identifier:xml:schema:1"
xmlns:autoid="http://www.autoidcenter.org/2003/xml"
xmlns="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified" version="1.0">
<annotation>
<documentation>
<documentation>
<autoid:copyright>Copyright )2003 Auto-ID Center, All Rights Reserved.
</autoid:copyright>
<autoid:disclaimer>Auto-ID Center, its members, officers, directors, employees,
or agents shall not be liable for any injury, loss, damages, financial or
otherwise, arising from, related to, or caused by the use of this
document. The use of said document shall constitute your express consent to the
foregoing
exculpation.</autoid:disclaimer>
<autoid:program>Auto-ID version 1.0</autoid:program>
<autoid:purpose>PML Core Specification version 1.0</autoid:purpose>
</documentation>
</documentation>
</annotation>
<element name="ID" type="pmluid:IdentifierType"/>

```

```
<annotation>
<documentation>
<autoid:definition>A reusable element of type 'IdentifierType'
</autoid:definition>
</documentation>
</annotation>
<complexType name="IdentifierType">
<annotation>
<documentation>
<autoid:definition>A character string to identify and distinguish uniquely, one
instance of an object
in an identification scheme from all other objects within the same scheme
</autoid:definition>
</documentation>
</annotation>
<simpleContent>
<extension base="token">
<attribute name="schemeID" type="token" use="optional">
<annotation>
<documentation>
<autoid:definition>The identifier of the identification scheme
</autoid:definition>
</documentation>
</annotation>
</attribute>
<attribute name="schemeAgencyID" type="token" use="optional">
<annotation>
<documentation>
<autoid:definition>The identifier of the agency that maintains the
identification
scheme</autoid:definition>
</documentation>
</annotation>
</attribute>
<attribute name="schemeVersionID" type="token" use="optional">
<annotation>
<documentation>
<autoid:definition>The version of the identification scheme</autoid:definition>
</documentation>
</annotation>
</attribute>
<attribute name="schemeURI" type="anyURI" use="optional">
<annotation>
<documentation>
<autoid:definition>The Uniform Resource Identifier that identifies where the
Identification Scheme is located</autoid:definition>
</documentation>
</annotation>
</attribute>
</extension>
</simpleContent>
```

```
</complexType>
</schema>
```

---

## B.2. RFIDLocator Reader's Configuration Formalism

The Schema provided in this Appendix aims to describe the settings of the environment for the RFIDLocator. It is commented in further details in Subsection 5.7.

### B.2.1. RFIDLocator.xsd

---

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSpy v2005 U (http://www.xmlspy.com) by Diuf_group (DIUF) -->
<!-- Part of the RFIDLocator: www.gmipsoft.com/rfid Created by: D.Guinard -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
<xs:element name="RFIDLocator">
<xs:complexType>
<xs:sequence maxOccurs="unbounded">
<xs:element name="Reader">
<xs:complexType>
<xs:sequence maxOccurs="unbounded">
<xs:element name="LogicalAntenna">
<xs:complexType>
<xs:sequence>
  <xs:element name="Solver">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="JNDIENCName"
          type="xs:string"/>
        <xs:element name="MaxReadingTime"
          type="xs:int"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="Location">
    <xs:complexType>
      <xs:all>
        <xs:element name="Description"
          type="xs:string" minOccurs="0"/>
        <xs:element name="BusinessLocationNumber"
          type="xs:string"/>
      </xs:all>
    </xs:complexType>
  </xs:element>
  <xs:element name="PhysicalAntenna" maxOccurs="unbounded">
    <xs:complexType>
```

```
        <xs:sequence>
            <xs:element name="Action" type="xs:string"/>
        </xs:sequence>
        <xs:attribute name="id" type="xs:string"
            use="required"/>
    </xs:complexType>
</xs:element>
</xs:sequence>
<xs:attribute name="fiability" type="xs:string" use="optional"/>
</xs:complexType>
</xs:element>
</xs:sequence>
<xs:attribute name="model" type="xs:string" use="optional"/>
<xs:attribute name="brand" type="xs:string" use="optional"/>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
```

---

# C

## CD-ROM

On the CD-ROM of the project you find:

- The source code, Ant files and compiled binaries of the RFIDLocator.
- The APIs of the RFIDLocator.
- The binaries and sources of this documentation.
- Various documents that were of great use during this Bachelor Thesis.

Figures C.1 and C.2 provide a tree view of the CD-ROM's structure.

The content of the CD-ROM can also be downloaded from the official website of the project (see Appendix D). However, since some material contained on the media is subject to commercial licenses it was removed from the online version.

---

```

|-- api
|   |-- javadoc_rfidlocator           \\API of the application.
|   '-- javadoc_sun_rfid_software
|-- bachelor_thesis                   \\Binaries (pdf, ps, etc.) and
|   |-- appendix                       \\sources of this document
|   |-- codes                           \\and of the presentation.
|   |-- figures                         \\Sources of the figures.
|   '-- parts
|       |-- introduction_to_RFID
|       '-- towards_RFIDLocator
|-- config
|   '-- Event_Manager                 \\Files to configure the EM.
|-- documentation                     \\Manuals of the hardware
|   |-- Moxa                           \\components.
|   |-- Sun
|   |-- Tagsys
|   '-- UltraEnterprise
|-- documents                          \\Articles, white-papers, websites.
|-- log_book                           \\Hints, notes and remarks about
|-- software                            \\this work.
|   |-- RFIDLocator
|   |   |-- JMSBridge                 \\Bridge to install on the EM.
|   |   '-- RFIDLocatorEJB           \\Final application.
|   |       |-- build
|   |       |-- dist                  \\Deployable binary (.ear).
|   |       |-- lib                    \\Required libraries.
|   |       |-- src
|   |           |-- conf
|   |           |-- java
|   |           |   |-- ch
|   |           |       '-- unifr
|   |           |           '-- diuf
|   |           |               |-- RFIDLocator
|   |           |                   |-- ejb
|   |           |                       |-- delegate
|   |           |                           |-- entity
|   |           |                               |-- mdb
|   |           |                                   |-- sequence
|   |           |                                       |-- session
|   |           |                                           '-- solver
|   |           |                                               |-- to
|   |           |                                                   '-- util
|   |           |                                                       '-- serviceLocator
|   |           |   '-- meta-inf
|   |           |-- sql                 \\Tables and test sets.
|   |           '-- web                  \\JSP GUI.
|   |               |-- css
|   |               '-- images
|...|...   '-- xml                     \\Configuration formalism.

```

---

Figure C.1.: Tree view of the CD-ROM's content (first part)

---

```

| |-- Sun                                //Contains the software delivered
| | |                                    //by Sun (EM, AS, Adapters).
| | |-- current_lx_adapter //MedioL100 Adapter for the EM v1.
| | '-- new_lx_adapter    //MedioL100 Adapter for the EM v2.
|-- Tagsys                        //Contains the software delivered
| |-- DLLs                        //by Tagsys.
| | |-- Library Application DLL
| | | |-- Dlls
| | | |-- Doc
| | | '-- Example
| | | |-- Executable
| | | '-- Sources
| | | |-- TagsysLibrary_Test
| | | |-- res
| | |-- Medio STX DLLs //Binaries related to the Medio
| | | |-- Doc //L100 RFID reader.
| | | |-- PC
| | | |-- Dlls
| | | '-- Sample Projects
| | | |-- Medio Lx00 Example
| | | | |-- Executable
| | | | '-- Sources
| | | | |-- LX00_Example
| | | | |-- Release
| | | | |-- res
| | | |-- Medio S00x Example
| | | | |-- Executable
| | | | '-- Sources
| | | | |-- S00X_Example
| | | | |-- Release
| | | | |-- res
| |-- GemCore Loader //Embedded OS of the MedioL100.
| | |-- Doc
| | |-- Installer
| |-- Gemcore_STX_Application
| |-- L200 Explorer
| |-- Product Documentation
'-- work_report //Contains a summary of the
                //work reports regarding the
                //project.

```

---

Figure C.2.: Tree view of the CD-ROM's content (second part)

# D

## Website of the Project

A web-page was created for this project: <http://www.gmipsoft.com/rfid><sup>1</sup>. On this page you will find:

- The API of the project.
- The Binaries and sources of this documentation.
- The Binaries and sources of the RFIDLocator application (including the JMSBridge).
- Finally, the content of the CD-ROM can also be downloaded from this source (see Appendix C).

Figure D.1 provides a screenshot of this website.

---

<sup>1</sup>This URL is a shortcut to <http://diuf.unifr.ch/softeng/student-projects/completed/guinard/index.html>

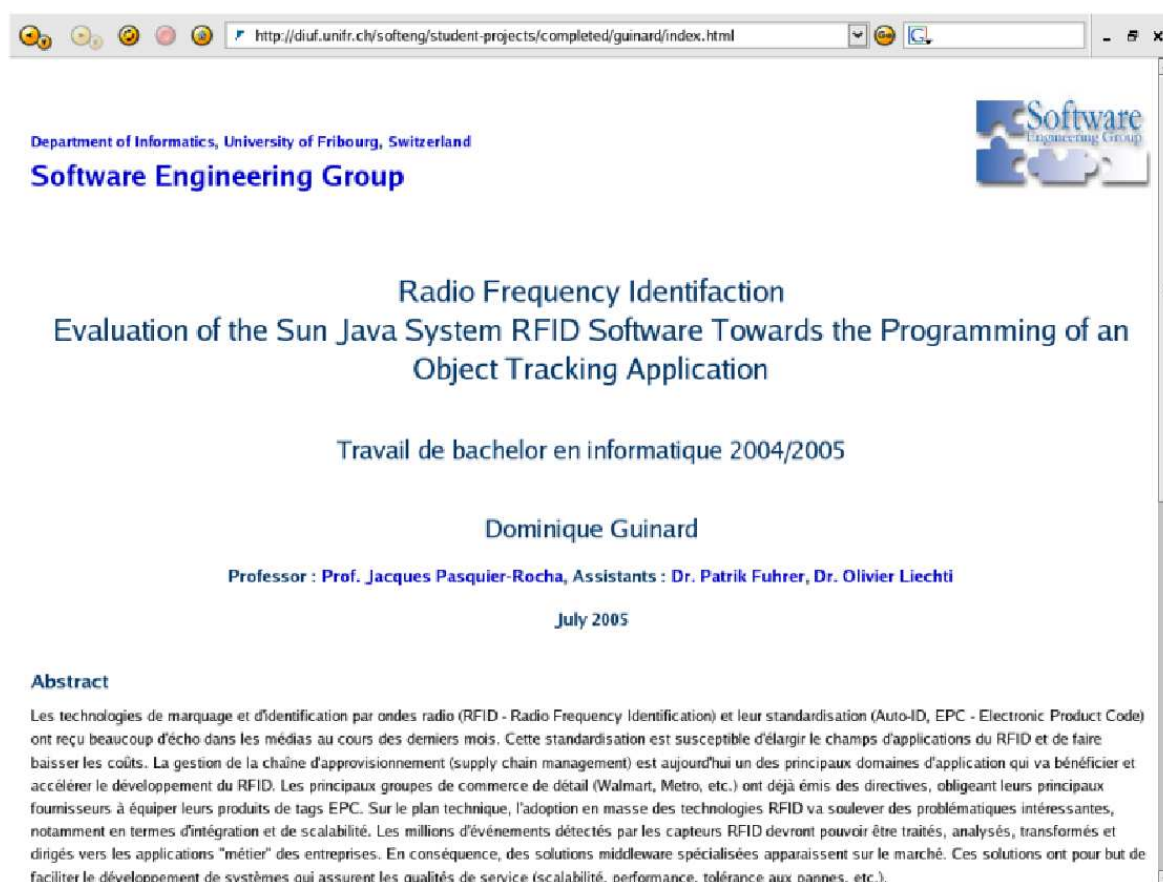


Figure D.1.: Screenshot of the project's official web-page

# E

## Hardware and Software

This appendix summarizes the hardware and software used for this Bachelor Thesis.

### E.1. Software

Description	Version	OS	Use
Kile [19]	1.6.3 and 1.8.1	Linux	IDE for the writing the documentation in $\text{\LaTeX}$ .
Dia [5]	0.94	Linux	Used to draw the figures and diagrams of the documentation.
Netbeans IDE [23]	4 and 4.1	Linux	IDE for programming the Java part of the application.
Borland Together [38]	Architect	Linux	Drawing the UML sequence diagrams.
Visual Paradigm for UML [39]	3	Linux	Drawing the UML use case and class diagrams.
XMLSpy [32]	v2005	Win XP	Editor for the XML Schema and XML files.
Sun Java System RFID Software [34]	v1	Solaris 9	Savant's implementation used to manage the RFID readers.
Apache Ant [1]	1.6.2	Generic	Builder for the distribution and sources.
XDoclet [42]	xdoclet-1.2.2 and 1.2.3	Generic	Generation of the EJB interfaces
JAXB RI [15]	1.0.4	Linux	Generation of the XML Data Binding classes.
Sun Java System Application Server [33]	Platform Edition 8.1	Linux	J2EE compliant AS holding the final application (the RFIDLocator).

Description	Version	OS	Use
Sun Java System Message Queue [33]	3, 2005Q1 (bundled with the AS)	Linux	JMS broker used to send and receive the PML messages on the EM and the AS.
The GIMP [10]	2.2	Linux	Image editing.
Dreamweaver [20]	MX 2005	Linux	GUI design and JSP templates.
Fireworks [20]	MX 2005	Linux	Design of the RFIDLocator 's logo.
L200 Explorer [37]	2004	Win XP	Writing EPCs on the RFID tags.

## E.2. Hardware

Description	Specificities	OS	Use
IBM ThinkPad R40	CPU: 1.3 GHz (Intel Pentium M) with 1 MB of cache. RAM: 768 MB	Fedora Core 2 and 4, Microsoft Windows XP	This machine was used for the developing of the application. During the tests it was also used as the machine holding the Application Server and the final application.
Ultra Enterprise 450	CPU: 4x UltraSPARC II 300MHz with 2 MB of cache each. RAM: 8 GB	Solaris 9	This server holds the Sun Java System Event Manager (Savant's implementation).
Moxa DE-311	2 Ports: 1 RS-232, 1 RJ-45	n.a.	Serial to Ethernet conversion of the events reported by the Medio RFID reader.
Medio L100	Power: 100 to 4000 MHz. Antennae: 2, Range: reading up to 30 cm	GemCoreOS	Physical RFID reader used for the tests.

# F

## Licenses of the Project

### F.1. License of this Document

Copyright (c) 2005 Dominique GUINARD.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.

The GNU Free Documentation License can be read from [8].

### F.2. License of the RFIDLocator and the JMSBridge

The RFIDLocator: An Assets Tracking Application using the RFID.

JMSBridge: A JMS relay Application

Copyright (C) 2005 Dominique GUINARD

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

The GNU General Public License can be read from [11].

#### Important Note

The RFIDLocator and the JMSBridge are released under the GNU GPL however, some of the libraries **they use** (sax.jax, j2ee.jar) **were not released under GNU GPL** licenses. All these libraries are located in `software/RFIDLocator/JMSBridge/lib` and `software/RFIDLocator/RFIDLocatorEJB/lib`.

# Bibliography

- [ACM03] Deepak Alur, John Crupi, and Dan Malks. *Core J2EE Patterns*. Sun Microsystems Press, second edition, 2003.
- [ACM05] Special Issue: RFID: tagging the world. *Communications of the ACM*, 48, September 2005.
- [Ada] Russ Adams. BarCode 1: a quick tour FAQ. Website. <http://www.adams1.com/pub/russadam/barcode1.html> (accessed November 21, 2004).
- [Aut] Auto-ID Labs. Website. <http://www.autoidlabs.org/aboutthelabs.html> (accessed December 21, 2004).
- [Aut04] Auto-ID Labs, Cambridge, UK. *The EPC information service (EPCIS)*, 2004.
- [BMKL01] David L. Brock, Timothy P. Milne, Yun Y. Kang, and Brendon Lewis. White Paper: The Physical Markup Language. Technical report, Auto-ID, 2001.
- [Bro01] David L. Brock. White Paper: The Electronic Product Code (EPC). Technical report, Auto-ID , 2001.
- [Col04] Jonathan Collins. IBM launches RFID middleware. *RFID Journal (online)*, 2004. [Retrieved December 21, 2004, from <http://www.rfidjournal.com/article/articleview/1291/1/1/>].
- [EPC04] EPCglobal. EPC Tag Data Standards Version 1.1 Rev.1.24. Technical report, EPC-global, 2004.
- [FAOH03] Christian Floerkemeier, Dipan Anarkat, Ted Osinski, and Mark Harrison. PML Core Specification 1.0. Technical report, Auto-ID Center, 2003.
- [Fin03] Klaux Finkenzeller. *RFID Handbook*. John Wiley & Son, second edition, 2003.
- [Ger00] Neil Gershenfeld. *When things start to think*. Owl Books, 2000.
- [Gor04] Dirk Gorissen. Getting Your Computer to Make Coffee. Website, 2004. [Retrieved August 1, 2005, from <http://dirk.campinaria.be/uni/papers/thesisPaper.pdf>].
- [Goy03] Amit Goyal. Technical Report: Savant Guide. Technical report, Auto-ID Center, 2003. [Retrieved December 30, 2004, from <http://www.autoidlabs.org/whitepapers/mit-autoid-tr015.pdf>].
- [GS04] Alka Gupta and Mayank Srivstava. Developing Auto-ID Solutions using Sun Java System RFID Software. Technical report, Sun Microsystems, 2004.

- [Har] Mark Harrison. EPC Tag Data Translation. Website. <http://www.autoidlabs.org/Cambridge/TDS/> (accessed December 30, 2004).
- [HM02] Elliotte Rusty Harold and W. Scott Means. *XML in a Nushell*. O'Reilly, second edition, 2002.
- [HMSS03] Uwe Hansmann, Lothar Merk, Martin S., and Nicklous Thomas Stober. *Pervasive Computing*. Springer, second edition, 2003.
- [Hug05] Christoph Hugenschmidt. SAP und Migros lancieren RFID-Diskussion in der Schweiz. *Inside-it.ch*, 2005. [Retrieved July 27, 2005, from <http://tinyurl.com/9bfff>].
- [Mah04] Qusay H. Mahmoud. Getting Started with Java Message Service (JMS). Technical report, Sun Microsystems, 2004. [Retrieved December 27, 2004, from <http://java.sun.com/developer/technicalArticles/Ecommerce/jms/>].
- [McL02] Brett McLaughlin. *Building Java Enterprise Applications: Volume 1: Architecture*. O'Reilly, first edition, 2002.
- [Mea04] Michael Mealling. EPCglobal Object Name Service (ONS) 1.0. Technical report, EPCglobal, 2004. Working Draft Version of April 15.
- [MH04] Richard Monson-Haefel. *Enterprise JavaBeans*. O'Reilly, fourth edition, 2004.
- [Mor] Roland Moreno. Website. [http://fr.wikipedia.org/wiki/Roland\\_Moreno](http://fr.wikipedia.org/wiki/Roland_Moreno) (accessed May 05, 2005).
- [O'C05a] Mary Catherine O'Connor. Calif. Senate Approves RFID Bill. *RFID Journal (online)*, 2005. [Retrieved August 1, 2005, from <http://www.rfidjournal.com/>].
- [O'C05b] Mary Catherine O'Connor. RFID and the Media Revolution. *RFID Journal (online)*, 2005. [Retrieved July 27, 2005, from <http://www.rfidjournal.com/article/articleview/1508/1/1/>].
- [O'C05c] Mary Catherine O'Connor. Survey Reveal Dubious Consumers. *RFID Journal (online)*, 2005. [Retrieved July 26, 2005, from <http://www.rfidjournal.com/article/articleview/1409/-1/1/>].
- [Ort04] Ed Ort. Ease of Development in Enterprise JavaBeans Technology. *Sun Developer Network*, 2004. [Retrieved July 19, 2005, from <http://java.sun.com/developer/technicalArticles/ebeans/ejbease/>].
- [Rob04] Mark Roberti. EPCglobal Ratifies Gen 2 Standard. *RFID Journal (online)*, 2004. [Retrieved December 21, 2004, from <http://www.rfidjournal.com/article/articleview/1293/1/1/>].
- [Rob05] Mark Roberti. Delta to Test Tags on Aircraft Engines. *RFID Journal (online)*, 2005. [Retrieved May 05, 2005, from <http://www.rfidjournal.com/article/articleview/1477/1/1/>].
- [Sch05] Thomas Schnee. Puce Espionne à surveiller de près. *La Liberté*, 2005.
- [Sco] Erica Scott. Law Firm Finds Relief in Automated File Tracking. Website. [Retrieved March 17, 2005, from <http://cms.3m.com/cms/US/en/2-115/kurRIFV/view.jhtml>].
- [SFV04] Frank Siegemund, Christiand Floerkemeier, and Harald Vogt. The Value of Handhelds in Smart Environments. In *Organic and Pervasive Computing - ARCS 2004*, pages 291–308. Springer, 2004.

- [She04] Steven Shepard. *RFID Radio Frequency Identification*. McGraw-Hill Professional, 2004.
- [Sta] Beat Stadtmann. RFID: Schnell gelesen, richtig ausgewertet. Intranet Website.
- [Sun] Sun ONE Administration Console Tutorial. Website. <http://docs.sun.com/source/816-5922-10/tutorial.html> (accessed December 27, 2004).
- [Sun97] Sun Microsystems, Inc. *Ultra Enterprise 450 Server Owner's Guide*, 1997. [Retrieved July 07, 2005, from [http://www.sun.com/products-n-solutions/hardware/docs/Servers/Workgroup\\_Servers/Sun\\_Enterprise\\_450/](http://www.sun.com/products-n-solutions/hardware/docs/Servers/Workgroup_Servers/Sun_Enterprise_450/)].
- [Sun04a] Sun Microsystems, Inc. *RFID Software Administration Guide*, 2004. [Retrieved October 25, 2004, from <http://docs.sun.com>].
- [Sun04b] Sun Microsystems, Inc. *RFID Software Developer's Guide*, 2004. [Retrieved October 25, 2004, from <http://docs.sun.com>].
- [Sun04c] Sun Microsystems, Inc. *RFID Software Installation Guide*, 2004. [Retrieved October 25, 2004, from <http://docs.sun.com>].
- [Sun04d] Sun Microsystems, Inc. *Tagsys Adapter v1.0*, 2004.
- [Wik] RFID from Wikipedia, the free encyclopedia. Website. <http://en.wikipedia.org/wiki/RFID> (accessed November 22, 2004).

Note: A copy the most important documents cited in this Bachelor Thesis is available on the CD-ROM of this work (see Appendix C).

# Referenced Web Ressources

- [1] Download page of Apache Ant. <http://ant.apache.org/bindownload.cgi> (accessed July 30, 2005).
- [2] Online Manual for Apache Ant. <http://ant.apache.org/manual/index.html> (accessed July 31, 2005).
- [3] Online version of the RFIDLocator's API. <http://www.gmipsoft.com/rfid/api> (accessed July 30, 2005).
- [4] Auto-ID Labs. <http://www.autoidlabs.org> (accessed July 28, 2005).
- [5] Homepage of the Dia project. <http://www.gnome.org/projects/dia> (accessed August 16, 2005).
- [6] Microsoft .NET. <http://www.microsoft.com/net> (accessed July 28, 2005).
- [7] EPCglobal Inc. <http://www.epcglobalinc.org> (accessed July 30, 2005).
- [8] Free Documentation Licence (GNU FDL). <http://www.gnu.org/licenses/fdl.txt> (accessed July 30, 2005).
- [9] Apache Geronimo (Application Server). <http://geronimo.apache.org> (accessed July 29, 2005).
- [10] The GIMP. [www.http://www.gimp.org](http://www.gimp.org) (accessed August 17, 2005).
- [11] General Public License (GNU GPL). <http://www.gnu.org/licenses/gpl.txt> (accessed September 14, 2005).
- [12] Online Manual for Apache Ant. <http://struts.apache.org> (accessed July 31, 2005).
- [13] Hôpital Universitaire de Genève. <http://www.hug-ge.ch> (accessed July 28, 2005).
- [14] J2EE. <http://java.sun.com/j2ee> (accessed July 29, 2005).
- [15] Download page of the Java Web Service Developer Pack. <http://java.sun.com/webservices/downloads/webservicespack.html> (accessed July 29, 2005).
- [16] JBoss (Application Server). <http://www.jboss.org> (accessed July 29, 2005).
- [17] Official homepage of the JINI Technology. <http://www.sun.com/software/jini/> (accessed August 1, 2005).
- [18] Jonas (Application Server). <http://jonas.objectweb.org> (accessed July 29, 2005).
- [19] KDE Integrated LaTeX Environment. <http://kile.sourceforge.net> (accessed August 15, 2005).
- [20] Macromedia Software. [www.http://www.macromedia.com](http://www.macromedia.com) (accessed August 17, 2005).

- 
- [21] Moxa. [www.http://www.moxa.com](http://www.moxa.com) (accessed July 28, 2005).
  - [22] Download page of MySQL (Community Licence). <http://dev.mysql.com/downloads/> (accessed July 29, 2005).
  - [23] Home of the Netbeans IDE. <http://www.netbeans.org> (accessed July 31, 2005).
  - [24] Download page of Oracle. <http://www.oracle.com/technology/software/index.html> (accessed July 29, 2005).
  - [25] Philips. [www.http://www.philips.com](http://www.philips.com) (accessed July 28, 2005).
  - [26] PML website of the Auto-ID center. <http://web.mit.edu/mecheng/pml/> (accessed August 1, 2005).
  - [27] Download page of PostGreSQL. <http://www.postgresql.org/download> (accessed July 29, 2005).
  - [28] UPM Rafsec. <http://www.rafsec.com> (accessed July 28, 2005).
  - [29] Official Website of the RFIDLocator project. <http://www.gmipsoft.com/rfid> (accessed July 30, 2005).
  - [30] Website of the RFIDLocator project. <http://www.gmipsoft.com/rfid> (accessed July 30, 2005).
  - [31] Homepage of the RIO project. <http://rio.jini.org> (accessed August 1, 2005).
  - [32] XML Spy. <http://www.xmlspy.com> (accessed August 15, 2005).
  - [33] Download page of the Sun Java System Application Server (Platform Edition 8.1). <http://java.sun.com/j2ee/1.4/download.html> (accessed July 25, 2005).
  - [34] Website of the Sun Java System RFID Software . <http://www.sun.com/software/solutions/rfid/> (accessed July 28, 2005).
  - [35] SUN Microsystems Switzerland. <http://ch.sun.com> (accessed August 13, 2005).
  - [36] Tag data translation prototype. <http://www.scenicviews.org/AutoID/TDT/> (accessed July 28, 2005).
  - [37] Tagsys Corp. <http://www.tagsys.net> (accessed July 28, 2005).
  - [38] Borland Together. [http://www.borland.com/us/products/core\\_architect/](http://www.borland.com/us/products/core_architect/) (accessed August 16, 2005).
  - [39] Visual Paradigm. <http://www.visual-paradigm.com/> (accessed August 16, 2005).
  - [40] World Wide Web Consortium: Naming and Addressing. [www.http://www.w3.org/Addressing](http://www.w3.org/Addressing) (accessed September 13, 2005).
  - [41] IBM Websphere (Application Server). <http://www-306.ibm.com/software/websphere/> (accessed July 29, 2005).
  - [42] XDoclet. <http://xdoclet.sourceforge.net> (accessed July 29, 2005).

# Index

- .NET, 66
- Auto-ID , 5, 13, 14
- EPCglobal , 14
- RFIDLocator
  - user manual, 75
- Action, 50
- Apache Geronimo, 67
- Application Server, 71
- Asynchronous processing, 66
- Autodeploy, 89
- Automatic Identification, 5
- Barcodes, 5
- Barcodes, next generation, 11
- BufferedObservation, 52
- CASPIAN, 93
- Comparison of Auto-ID systems, 11
- Concurrent requests, 66
- Contact less, 5, 11
- Content Object, 55
- Datacenter, 37
- Definitions, 37
- Deploying, 89
- Design patterns, 60
- Detach, 78
- Distributed Application, 65
- EAN, 6
- EJB, 66
- Electronic Product Code, 15
- ENC, 47
- Enterprise JavaBeans, 66
- Entities, 47
- Entity Beans, 47
- EPC, 15, 60
  - conversion engine, 20
  - Encoding schemes, 16
  - Grammar, 19
  - identifier, 15
  - manager, 16
  - manufacturer, 16
  - tag, 15, 71
  - tag data translation, 20
  - transponder, 15
  - URI, 17
    - for EPC Tags, 18
    - for invalid tags, 18
    - patterns, 19
    - pure entities, 18
- ERP, 65
- European Article Number, 6
- Event Manager, 65, 69, 82
- FDL, 112
- GPL, 112
- Hospital, 37
- Identity Concept, 19
- Internet, 14
- J2EE design patterns, 60
- Java Message Service, 66
- JAXB, 53
- JBoss, 67
- JDOM, 55
- JMS, 66
- JMS Bridge, 88
- JNDI, 47
- Jonas, 67
- JWSDP, 53
- Labels, 11
- Law firm, 36
- License, 112
- Linux, 82

- LocationManager, 53
- LocatorObservation, 51
- Manual, 75
- Marshalling, 53
- MDB, 56, 66
- Medio L100, 69, 83
- Message Driven Bean, 56, 66
- Microchip, 8
- Moxa, 69
- Museum, 37
- MySQL, 67
- NAPTR, 32
- Object Model, 47
- Observation, 51
- ObservationManager, 56
- OCR, 7
- Optical character recognition, 7
- Oracle, 67
- Patterns, 60
- Pervasive, 12
- Philips, 71
- PML, 21
  - core, 21
  - specification, 21
  - XML schema, 21, 97
- PML Reader, 82
- PML Simulator, 79
- PMLSimulator, 55
- PointBase, 67
- PostGreSQL, 67
- Primary Keys, 60
- Rafsec, 71
- Reader's Configuration, 62
- ReaderManager, 53, 62
- RFID, 8
  - advantages, 10
  - chips, 10
  - definition, 5
  - Hardware architecture, 8
  - housing, 12
  - labels, 11
  - Overview of the system, 9
  - power-supply, 11
  - Radio Frequency Identification, 8
  - reader, 83
  - transponder, 10
- RFIDLocator
  - attach, 78
  - configure the Environment, 77
  - find, 79
  - GUI, 75
  - install guide, 82
  - interface, 75
  - license, 112
  - manual, 75
  - new user, 77
  - PML Simulator, 79
  - seek, 79
  - starting the application, 90
  - test set, 79
- ROI, 93
- SAP, 65
- Savant, 24
  - adapters, 26
  - EPCIS, 24
  - Event Manager, 9, 26
  - filters, 26
  - loggers, 30
  - RIED, 24
  - TMS, 24
- SensorListener, 56
- Sequence, 60
- Sequencer, 60
- Serial to TCP/IP, 69
- Services, 52
- Session Beans, 52
- Session Façade, 52
- Sessions, 52
- Smart label, 71
- Smartcards, 7
- Software Architecture, 46
- Solaris, 82
- Solver
  - architecture, 58
  - concrete solver, 57
  - different actions solver, 58
  - observation solver interface, 58
  - single type, 57
- Solvers, 56
- Solving algorithms, 56
- Sparc, 69
- Stop RFID, 93
- Sun, 69
- Sun Java System Application Server, 67
- Supply Chain, 93
- Supply chain, 21
- Tag, 10, 15, 71
- Tagsys, 69, 83

---

Terminology of the application, 37  
TraceableObject, 51  
TraceableObjectManager, 53  
Transponder, 8, 10, 71

Ubiquitous, 5, 12  
Ultra Enterprise 450, 69  
Ultrasparc, 69  
Universal Product Code, 6  
Unmarshalling, 53  
UPC, 6  
URI, 17  
URL, 17  
User, 51  
UserManager, 52

Vocabulary, 37

WebSphere, 67

XDoclet, 67  
XML, 62  
XML Data Binding, 53  
XML Schema, 62