

Department of Computer Science  
Pervasive and Artificial Intelligence Research Group  
University of Fribourg, Switzerland  
<http://diuf.unifr.ch/pai>

# Smart Badge:

A project in Ubiquitous Computing  
Project's Documentation and Report

Pedro De Almeida & Dominique Guinard  
February 2006



[dominique.guinard@unifr.ch](mailto:dominique.guinard@unifr.ch)  
<http://www.gmipsoft.com/unifr>

[pedro.dealmeida@unifr.ch](mailto:pedro.dealmeida@unifr.ch)  
<http://almeidap.hyperscheme.ch>



“All say, in essence, that only when things disappear in this way are we freed to use them without thinking and so to focus beyond them on new goals.”

- *Mark Weiser, 1988, [Wei88]*

## Abstract

First depicted by Mark Weiser in the late eighties, Ubiquitous Computing (aka UbiComp) seems to be among the mega-trends of this decade.

Developed in frame of the Master Course in Ubiquitous Computing at the University of Fribourg (Switzerland), the Smart Badge project mainly explores two trends which lead to the UbiComp era: “**better sensing technologies**” and “**invisible computing**”.

Indeed, the Smart Badge project is an infrastructure aiming to offer location-contextual services to the members of an institution (e.g. University students, employees in a corporate, etc.) or to the visitors of an event (e.g. conferences, exhibitions, shows, museums, etc.). To achieve this goal each human actor of the system is equipped with a **smart badge** containing a Radio Frequency Identification (RFID) transponder. Simultaneously, a number of interconnected computers and sensors trace the users within the predefined environment, providing her with contextual services.

While the use of RFID technologies permits an exploration of the “better sensing technologies”, the information is processed in the background as meant by the “invisible computing” trend.

This document describes the project from both a conceptual and an technical view-point. Besides, it offers a user and administrator guide for installing and using the various components of the Smart Badge infrastructure. Eventually, it provides an evaluation of the results as well as some thoughts about further extends and developments.

<b>Keywords:</b>
------------------

Ubiquitous Computing, RFID, Auto-ID, EPC, RFIDLocator, Mobile Services, Contextual Services, Kinetic User Interface, Handheld Devices.
--

# Contents

<b>1</b>	<b>Background</b>	<b>4</b>
1.1	Motivations and Goals . . . . .	4
1.2	State of the Art . . . . .	5
1.3	Use Cases . . . . .	6
1.3.1	Introduction . . . . .	6
1.3.2	Formalized Use Cases . . . . .	6
<b>2</b>	<b>Functional Description</b>	<b>8</b>
2.1	Requirements . . . . .	8
2.2	Layered Architecture . . . . .	9
2.2.1	Sensing Layer . . . . .	9
2.2.2	Environment Solving Layer . . . . .	11
2.2.3	Service Layer . . . . .	12
<b>3</b>	<b>Operational Instructions</b>	<b>16</b>
3.1	System Requirements . . . . .	16
3.1.1	Software . . . . .	16
3.1.2	Hardware . . . . .	17
3.2	User Guide . . . . .	17
3.2.1	Installation and Tutorials . . . . .	17
<b>4</b>	<b>Evaluation</b>	<b>25</b>
4.1	Adherence with the Specification . . . . .	25
4.2	Experiments . . . . .	25
4.3	Analysis of the Results . . . . .	26
4.3.1	Impressions . . . . .	26
4.3.2	Technical Limitations . . . . .	27
<b>5</b>	<b>Future Works</b>	<b>28</b>
5.1	Possible Extensions . . . . .	28
<b>6</b>	<b>Project Management</b>	<b>29</b>
6.1	Team Composition . . . . .	29
6.2	Individual contributions . . . . .	30

# List of Figures

1.1	Use cases of the Smart Badge project . . . . .	7
2.1	Smart Badge's global architecture . . . . .	8
2.2	Smart Badge layered architecture . . . . .	9
2.3	Hexadecimal representation of an GID-EPC encoded on 96 bits . . . . .	9
2.4	Components of the Event Manager Infrastructure . . . . .	10
2.5	Modules of the Event Management System . . . . .	11
2.6	XML formalism for Environment-solving to Service Layer communication . . . . .	12
2.7	Class diagram of the Smart Server . . . . .	12
2.8	The entities involved in the Service Management Layer . . . . .	14
2.9	User, editor and administrator capabilities . . . . .	15
2.10	User, editor and administrator capabilities . . . . .	15
3.1	Configuration of the physical environment . . . . .	19
3.2	Adding a new user to be traced . . . . .	20
3.3	Features available from the user control panel . . . . .	23
3.4	Features available from the editor control panel . . . . .	23
3.5	Features available from the admin control panel . . . . .	23
3.6	Listing of smart observations . . . . .	24
4.1	Services settings for the UGS use case . . . . .	26

# 1

## Background

---

1.1	Motivations and Goals . . . . .	4
1.2	State of the Art . . . . .	5
1.3	Use Cases . . . . .	6
1.3.1	Introduction . . . . .	6
1.3.2	Formalized Use Cases . . . . .	6

---

### 1.1 Motivations and Goals

To better understand the goal of the Smart Badge project and the idea of its foundations, let us describe the day of Armand, student at a University where the Smart Badge infrastructure was installed:

*Armand is a student in media and communication at the University of Fribourg. In the morning, he enters the University's main building. His cellphone vibrates: today's vegetarian meal served at the canteen appears on the display.*

*On the way to his management lecture, he stands in front of the major students' association to check the upcoming students' parties. While he reads it, a summary is sent to his email.*

*As Armand enters the lecture room, his cellphone goes silent after displaying a summary of the lesson. Before the professor begins with the lecture, he checks the names and subjects of the attendees and welcomes the class. Armand open up his laptop and starts taking note on the local version of the lecture slides he just received.*

This description is what we call a “Smart Day” understand: a day within an area able to sense and react on the observed events. Thus, the goal of the Smart Badge project is to build a framework<sup>1</sup> (both physical and logical), supporting location-sensing enabled environments. This framework should be able to transmit services on the behalf of the observations or, to be more precise, on the behalf of the users' displacements within the predefined area.

From a user's view-point the Smart-Badge project could be defined as **Kinetic User Interface** (or KUI). That is an interface using which the displacements, in terms of physical displacement within the environment, determine the action

---

<sup>1</sup>Unless clearly mentioned, the word “framework” should be taken in its vast sense rather than its purely software-engineering related meaning.

to be launched. It is worth noting we thought<sup>2</sup> of the KUI acronym as the end-user interface was neither a Graphical User Interface nor a Tangible User Interface.

Furthermore, our goal is to describe and implement an infrastructure fulfilling the needs of an application in real conditions. This is an important point as it drives most of the design and justifies some non-trivial choices for the overall architecture. In particular the described infrastructure should comply with the following general requirements:

1. Resist to high demand: As the institutions may range from small committees to large corporates the system should be scalable.
2. Deal with the heterogeneity among physical configurations, services and environments: As the Smart Badge is a framework rather than a simple application it needs to be as generic as possible.
3. Invisible Computing: In order to be ubiquitous, the infrastructure should be as pervasive as possible. All the computation should be moved to the background enabling the user to focus on the services' content, not on the method to get them.

While the first requirements are a consequence of our will to design the prototype of a real business application<sup>3</sup>, the last was largely inspired by Weiser's vision of ubiquitous computing summarized in [Wei88].

## 1.2 State of the Art

As Smart Badge is an infrastructure rather than a single application, it is composed of many applications involving a number of concepts. Thus, assembling the state of the art of Smart Badge is not an easy task as it requires to review a great amount of research topics. Anyhow, this section provides some hints about the state of the most related research topics and commercial applications.

At first one may notice that the main characteristics of the Smart Badge system is to offer **ubiquitous** services within a **predefined area**. Thus, it can be related to the researches around the smart homes. Among these we may cite the numerous projects from Microsoft Corp. ([Mic05]) or from IBM [IBM05]. Besides, the book [HMSS03] contains an interesting chapter on the subject. However, most of these papers and researches focus on the appliances which could make our homes smarter. This is not quite the view-point of the Smart Badge infrastructure which focuses on services related to the location of a user rather than on the equipment surrounding him.

Furthermore, Smart Badge mostly uses the **location of a user to contextualize the delivered services**. As a consequence researches in the domain of ubiquitous assets tracking are of great interest to this project. As assets tracking experienced a craze with the Radio Frequency Identification technologies, studies in this area are abundant. As a starting point one may consider [FGL06] which demonstrates the use of RFID for business tracking applications. Additionally, [Gui05] reports the design, implementation and uses of an enterprise application for assets localization. contains many pointers to ressources and paper on the

---

<sup>2</sup>As far as we know this term was never mentioned before in the User Interfaces related literature.

<sup>3</sup>Here, the term prototype of a "real business application" can be opposed to a laboratory prototype.

subject.

Finally, even if the Smart Badge project does not specify the devices to which the service delivery is eventually done, handheld appliances have, to date, the most interesting characteristics for such an ubiquitous application. Therefore studies related to the use of **handheld** devices in ubiquitous and pervasive environments are interesting for the Smart Badge infrastructure. In the domain, the article of Siegemund, Floerkemeier Vogt [SFV04] studies the value of handhelds in smart environments.

## 1.3 Use Cases

### 1.3.1 Introduction

The Smart Badge project does not comply with a finite set of use cases. Indeed, as mentioned before, it is basically a framework for providing location-contextual services to the members of an institution. The exact definition of both the words **location-contextual** and **services** is left to the institution using (and configuring) the Smart Badge infrastructure. It is not (or only partially) wired in the system. However, typical use cases arise. A sample of them is listed above:

- Providing information by SMS (Short Message Service) about upcoming conferences when entering a fair or an exhibition.
- Turning off cellular phones when entering a conference room.
- Offering the canteen's menu by SMS when entering a campus.
- Checking the amount of people present in a conference room.
- Checking the attendance of the students in a lecture room.
- Sending conference summaries by email to the attendees.
- Informing people by SMS about flight delays.
- Sending, by email the lecture's slides to students who **actually** attended the course.
- Building info-points people can approach to receive related informations by email or SMS.
- Sending "welcome" messages by SMS to people entering a company or an institution.
- Building "download-points" people can approach to upload dedicated applications on their machines (laptops, handhelds, desktops, etc.).
- etc.

### 1.3.2 Formalized Use Cases

To better understand the various uses of the Smart Badge project a formalism is required. Hence, Figure 1.1 presents the use cases in UML (Unified Modeling Language). The functionalities offered by the framework are separated into two sub-systems:

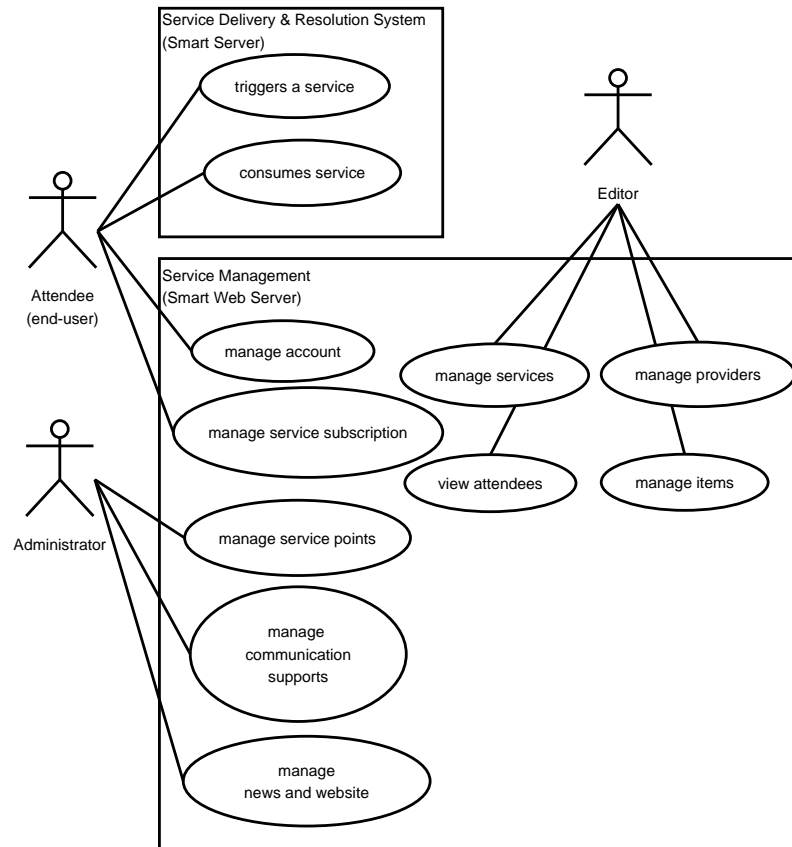


Figure 1.1: Use cases of the Smart Badge project

- The Service Delivery and Resolution System represents the algorithmic core layer. It is used by the end-user through what we call a KUI (Kinetic User Interface): the user moves within the predefined environment to “trigger services”. That is: the user’s displacements are converted into the delivery of 1... $n$  services by the Service Delivery and Resolution System.
- In turn, the Service Management System is the management layer. Interactions with this system follow a classical GUI (Graphical User Interface) scheme. Through this system the actors can manage and create the services delivered by the Service Delivery and Resolution System.

It is the purpose of the next chapter to define and specify more precisely the various components of these two sub-systems.

# 2

## Functional Description

---

<b>2.1</b>	<b>Requirements</b> . . . . .	<b>8</b>
<b>2.2</b>	<b>Layered Architecture</b> . . . . .	<b>9</b>
2.2.1	Sensing Layer . . . . .	9
2.2.2	Environment Solving Layer . . . . .	11
2.2.3	Service Layer . . . . .	12

---

### 2.1 Requirements

Based on a multi-tier infrastructure, the Smart Badge projet uses an complex architecture to achieve the goal of offering contextual services to users as introduced in Section 1.2. The requirements to deliver these services, from the initial state where an observation is intercepted to the state where the informations have been collected and sent to the corresponding users, are shown in Figure 2.1.

The **RFID Transponder**<sup>1</sup> is the identifier resource associated with an unique user. This resource can be activated by an **Sensor** so that identification data saved into the transponder can be read and forwarded to an **Event Manager**, which is in charge to transmit these observations to the **Application Server**. The tasks assured by the **Event Manager** are to get information data from a shared **database** and to send it to the user **device(s)**. Independently, the content of delivered services is managed via the **Web Interface**. This whole process of delivering services will be explained in the next section through the description of the different architecture layers.

<sup>1</sup>Radio Frequency Identification **T**ransmitter/**r**esponder

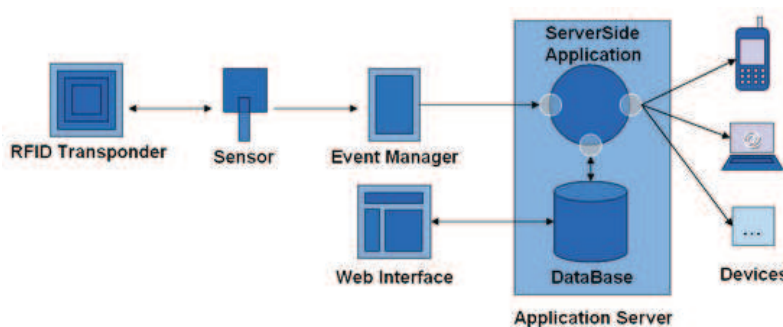


Figure 2.1: Smart Badge's global architecture

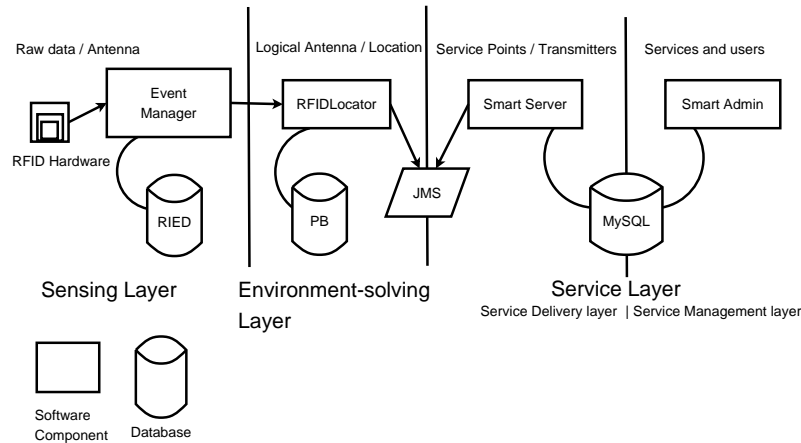


Figure 2.2: Smart Badge layered architecture

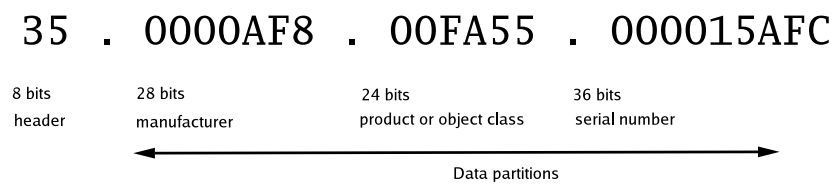


Figure 2.3: Hexadecimal representation of an GID-EPC encoded on 96 bits

## 2.2 Layered Architecture

Smart Badge architecture lies on three main layers, each providing a specific functionality (see Figure 2.2). As each layer owns his own database, we can assume that layers are independent and could be plugged into another application, using only his interface.

The first layer, called **Sensing Layer**, catches events and converts them into RFID observations. The next layer, **Environment Solving Layer** uses these observations to identify user location. As the **Service Layer** listens to the observations provided by the **Environment Solving Layer**, it will be able to deliver the service provided at the location where the user has been identified. This target is achieved by two sub-layers, the **Service Delivery Layer** and the **Service Management Layer**.

To better understand the steps of service delivering, the layers introduced before will be detailed in the next sections.

### 2.2.1 Sensing Layer

As said before, this is the lowest layer of the Smart Badge architecture. It is in charge of the management of the RFID readers (aka sensors) and of the processing of raw RFID events. This layer basically transforms the raw events into well-formatted RFID observations.

This layer should be as standardized as possible. Indeed, the adoption of approved standards at this point permits the use of the RFID infrastructure for other purposes such as security management or inter-organizational assets traceability. In particular the sensing layer of Smart Badge employs:

- The Electronic Product Code (EPC) (see Figure 2.3) as the unique identifier contained in the users' badges.

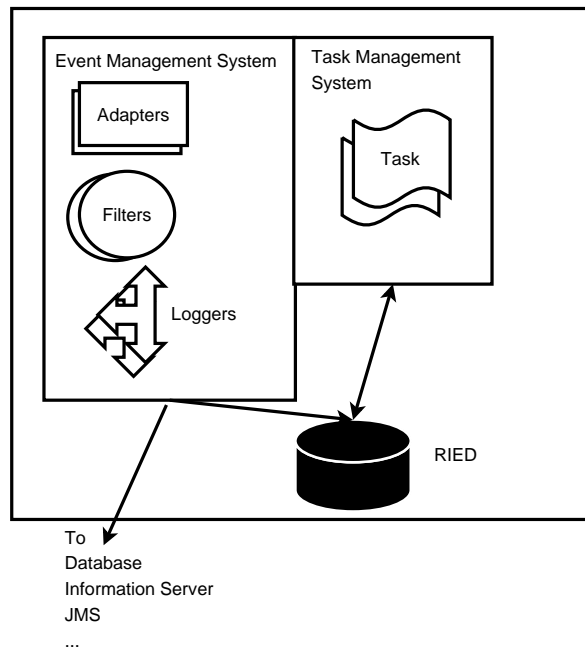


Figure 2.4: Components of the Event Manager Infrastructure

- The Physical Markup Language as an XML-based standard for information interchange between the Sensing Layer and the Environment-solving layer.
- An implementation of the RFID Event Manager as a standard for managing the RFID readers (see Figure 2.4). (This component will be further described below).

While these standards are important, understanding them is not crucial to appreciate the functionalities of Smart Badge. Thus, except for the Event Manager, no further details will be given here. However [FGL06], [Lie05] and [Gui05] provide a handfull of information and summaries on the subject.

### The Event Manager

The Event Manager (aka EM, formerly called “Savant”) is the main (software) component of the Sensing Layer. It is somehow like a tub of LEGO bricks, composed of many modules that are combinable to form an aggregate. The structure of the EM fits the Hollywood Principle<sup>2</sup>. This fact, among other properties, makes it a framework providing methods and tools to manage and capture RFID 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 appliances.
- **The Filters:** these modules are responsible for the dispatching or the trashing of the received events. Just about any type of filter can be designed. Technically speaking, it is worth noting that a filter has to implement the `EventFilterInterface`.
- **The Loggers:** are used to actually monitor or store the events. They are the end points of the EM’s chained structure.

<sup>2</sup>The Hollywood principle can be summarized by the sentence: “Do not call us, we will call you!”

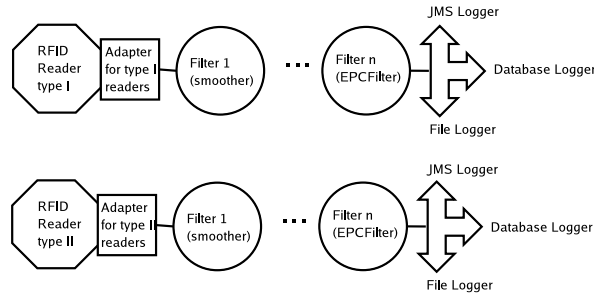


Figure 2.5: Modules of the Event Management System

On top of these elements, business applications can be quite easily built. Such software need no more to be aware of the tags' capture technical details. It just needs to implement a way to listen to a logger. This is basically what the components of the Environment-solving layer do and is the subject of the next subsection.

As a summary, Figure 2.5 gives a typical way of combining the modules of the Event Manager.

Eventually, it is worth noting that we did not implement an Event Manager for the Smart Badge project but rather installed and configured an implementation of the standard designed by Sun Microsystems. More information on the concrete product we used as the EM can be found in Section 3.1.1.

### 2.2.2 Environment Solving Layer

This second layer of the Smart Badge Infrastructure is in charge of tracking the users within the predefined area (e.g. a campus, conference rooms, etc.). Its core task is to transform the observations of RFID tags sent by the Sensing Layer over a JMS (Java Message Service) queue<sup>3</sup> into location tracking events. In short, it tells the next layer where the user is located based on the received RFID observations.

#### The Locator Component

In order to achieve its tracking goal the Environment Solving Layer needs a software component called “tracker” or “locator”. The locator needs to be aware of the environment and the readers composing it. In essence, it matches the antennae of the RFID sensors onto locations within the predefined area. In depth, however, it must be able to aggregate and analyze the received events in order to determine the most probable location corresponding to a series of RFID observations.

One of our requirements for the locator was to be able to capture the direction of the motion independently from the underlying type of RFID reader. This permits the Smart Badge system to offer a thinner granularity in regards to the contextualization of services. Indeed, this fact enables Smart Badge to deliver different services if one was entering a place or exiting it. It is worth noting at this point that this requirements could as well be fulfilled (with some significant efforts) by the Sensing Layer using the ALE (Application-Level Events) specification (see [Lie05]). Nevertheless a concrete locator developed at the University of Fribourg was chosen since it complies well with the needs of the Smart Badge project and offers additional capabilities for location tracking.

<sup>3</sup>See [Gui05] for a detailed description of the communication mechanisms between the Event Manager and the concrete locator (i.e. the RFIDLocator)

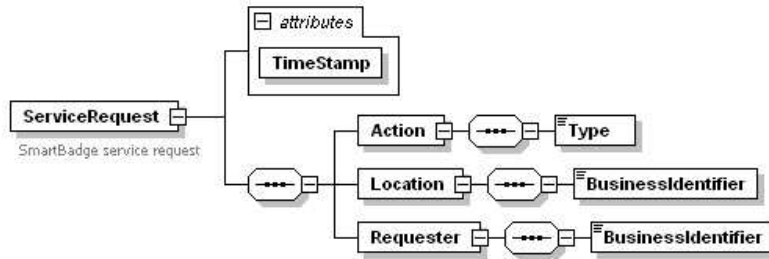


Figure 2.6: XML formalism for Environment-solving to Service Layer communication

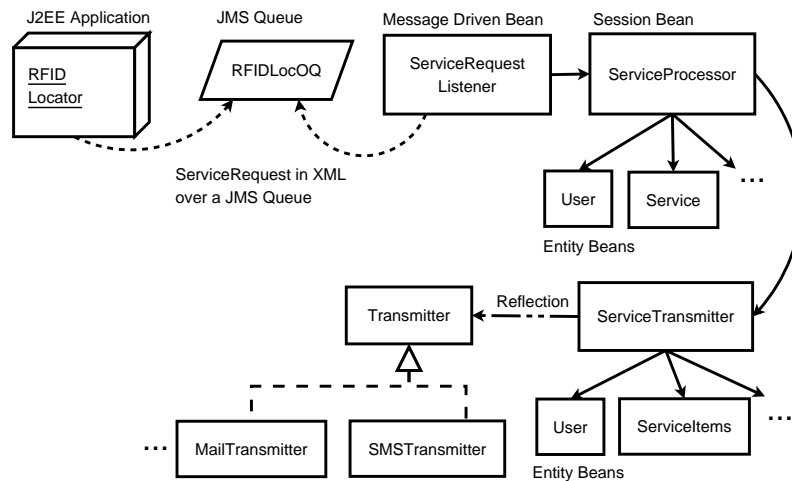


Figure 2.7: Class diagram of the Smart Server

Again, we did not implement a locator for the Smart Badge project but rather installed and configured the RFIDLocator mentioned above.

It is worth noting, however, that we extended the RFIDLocator by designing two new classes `SmartBadgeConcreteSolverSessionBean` and `SmartBadgeSingleTypeConcreteSolverSessionBean`. These components are used to convert the output of the RFIDLocator into a formalism which can be understood at the Service Delivery Layer. This formalism (the Smart Badge Service Request formalism) was designed using an XMLSchema (eXtended Markup Language) and is depicted on Figure 2.6. Basically, after transforming the RFID observation into locations' observation, the locator casts them into Smart Badge Service Requests and to the next layer using a JMS queue.

For more information about these components please consult the Javadoc API provided on the CD-ROM of the project.

### 2.2.3 Service Layer

The Service Layer is the core of the Smart Badge project. It is at this level that the Kinesthetic interactions of the users are going to be eventually transformed into concrete services (Service Delivery Layer, see 2.2.3). It is also the layer of Service Management where the actual services, providers and users are managed (Service Management Layer, see 2.2.3).

### Service Delivery Layer: The Smart Server

Located in the Service Delivery sub-layer, the Smart Server is in charge of:

1. Filtering Smart Badge Service Requests sent by the Environment Solving Layer.
2. Delivering the services.
3. Persisting service delivery acknowledgments in a a transacted database shared with the components of the Service Management Layer.

It is composed of numerous classes as of on the UML class-diagram shown on Figure 2.7. The following paragraphs will focus on a brief functional description of each class whereas a more technical view-point can be found in the Javadoc API of the project:

- **RFIDLocOQ**: This is the entry point of the Smart Server. It consists of a JMS Queue on which Service Requests are posted by the locator. In short, it is the interface between the Environment-solving Layer and the Service Layer.
- **ServiceRequestListener**: This component listens to the RFIDLocOQ queue. It catches the messages, parses them into **ServiceRequest** objects and forwards them to the **ServiceProcessor**.
- **ServiceProcessor**: This is the core algorithm of the Smart Server. It:
  - Checks whether the Service Requests are valid.
  - Contacts the **ServiceTransmitter** to actually deliver the service to be processed.
  - Persists services delivery acknowledgments in the form of **ServiceObservations**.

To achieve these goals it interacts with a number of other classes modeling the various objects of the Smart Server (such as **SbUserEB**, **SbServiceEB**, etc.).

- **ServiceTransmitter**: Called by the **ServiceProcessor**, this class is in charge of passing a service to a particular transmitter for final delivery. It is worth noting that the **ConcreteTransmitter** to be used is dynamically instantiated using the reflection mechanisms<sup>4</sup>. Again, to achieve its goal the **ServiceTransmitter** interacts with a number of other classes modeling the objects of the Smart Server (such as **SbSericItemsEB**, **SbUserEB**, etc.).

While the detailed architecture of these components is provided in the Javadoc API of the Smart Server, it is interesting to spend some time on the Transmitters architecture. Indeed, by decoupling the services from their concrete delivery method the Smart Server can be extended to new pervasive and handheld devices in a quite elegant manner. Actually, all one needs to do to offer a new delivery method is to write a class extending the **Transmitter** interface and (dynamically) load it in the application.

This way of programming the Smart Server was not very straightforward, however as the ubiquitous devices evolve quite fast, we thought this functionality as being mandatory for an extensible and viable architecture.

---

<sup>4</sup>In fact, a slightly modified version of the Java Reflection API which allows us to reflect on Enterprise Beans as well.

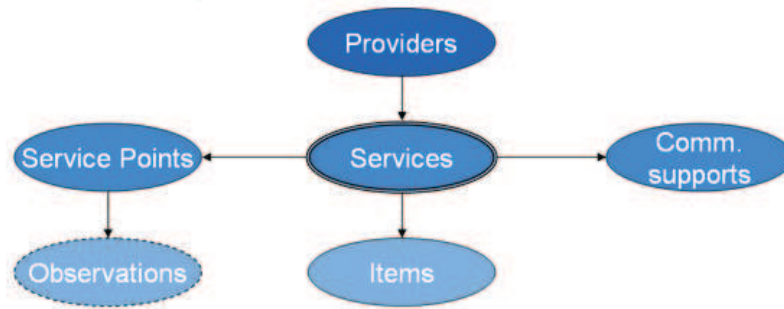


Figure 2.8: The entities involved in the Service Management Layer

### Service Management Layer: The Smart Web Server

The Service Management Layer offers the interface between the end-user and the Service Delivery layer. The main goal of this layer is to provide user-friendly features to set up an environment able to offer contextual information from desired locations to registered users.

In one abstract way, we can think of a set of entities acting together to deliver these services, as shown in Figure 2.8 and explained below:

- **Providers:** main entities of the environment, can provide one or more services.
- **Services:** set of information data intended for users, is attached to a service point and uses a communication support.
- **Service Points:** location where observations can be done (using a sensor generally), a service point can provide one or more services.
- **Observations:** events produced by an user localization in an area offering services (i.e. a service point)
- **Communication Supports:** method for sending information to the specified support (generally a device).
- **Items:** representation of the textual information, this is the output received by the user.

Furthermore, the Service Management Layer is also responsible for user management. An important aspect when talking about user management are roles and rights associated to these ones.

The configuration of the Smart Web server allows creation of three kind of users having different capabilities as depicted in Figure 2.9. See Section 3.2.1 to have a better look of these roles and how you can use them.

- **User:** this is the role for a standard account without special permissions. Nevertheless, Users should be able to create an account, edit their profile and subscribe or subscribe from **services**.
- **Editor:** this is an advanced role that permits to publish information. In addition of Users rights, Editors can also set up an contextual environment by managing providers, services and items. They are also authorized to view the observations provided by the **service points**.

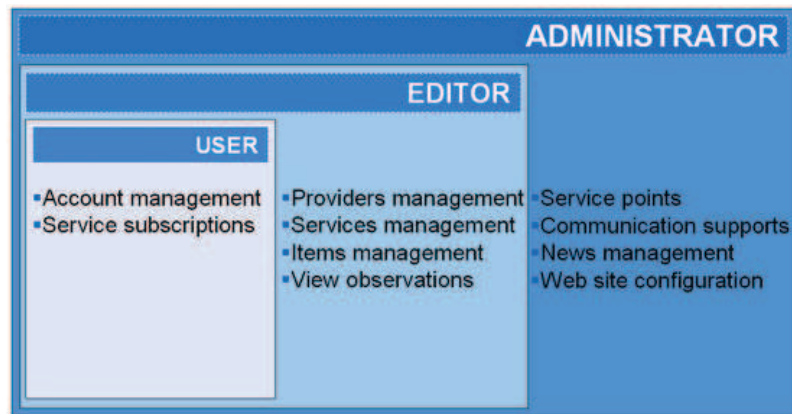


Figure 2.9: User, editor and administrator capabilities

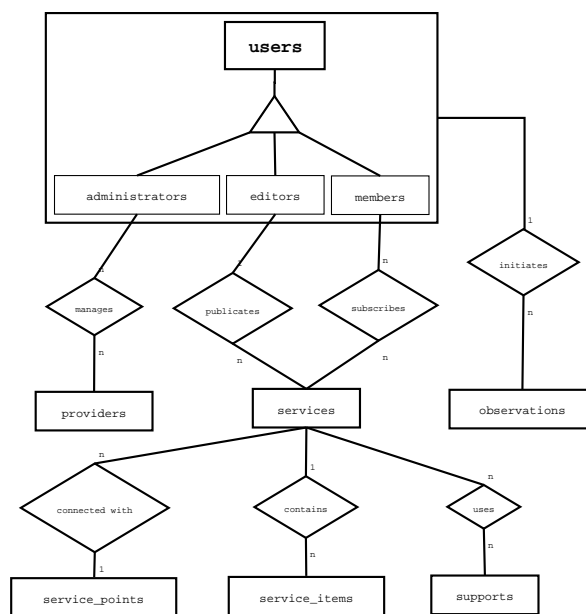


Figure 2.10: User, editor and administrator capabilities

- Administrator: this is the most important role which owns all capabilities. Except User and Editor rights, Administrator are also in charge to set up **service points**, define new communication supports, manage web site news and change the web site configuration.

Finally, all these constraints have been take in account and modelised by an **entity-relationship model** before exporting them to create the database structure. Figure 2.10 gives an overview about entities and their relations.

# 3

## Operational Instructions

---

<b>3.1 System Requirements</b> . . . . .	<b>16</b>
3.1.1 Software . . . . .	16
3.1.2 Hardware . . . . .	17
<b>3.2 User Guide</b> . . . . .	<b>17</b>
3.2.1 Installation and Tutorials . . . . .	17

---

### 3.1 System Requirements

#### 3.1.1 Software

After describing the functionalities of the Smart Badge project, we can now introduce the software used to achieve the implementation of a such infrastructure. As the global application consists on several software components, we will limit to only four of them but mentioning probably sub-components if necessary.

- **Event Manager:** this is the software component required for an application working with RFID and EPCs. Based on the notion of “service”, the Event Manager is well suited to network RFID readers and catch events produced by RFID tags. A RIED database is coupled with the Event Manager to store information about the events (see [Gui05] for more information).
- **Service Locator (J2EE):** this software component enables services location as described in Subsection 2.2.2. It uses a PointBase database to store observations and JMS as the communication API between the Environment Solving Layer and the Service Layer (see [Lie05] for more information).
- **Smart Server (J2EE):** this component handles the observations provided from the Service Locator and interacts with the user by sending him contextual information. The content of this information is persisted through EJB<sup>1</sup> from a MySQL database shared with the Smart Web Server.
- **Smart Web Server (PHP):** this last software component provides service management by giving to allowed users the ability to set up contextual information by creating items associated to services. The Smart Web Server communicates with the shared MySQL and manages the contents of services sent to users.

---

<sup>1</sup>Enterprise Java Beans

### 3.1.2 Hardware

If we talk about a multi-tier and distributed infrastructure, it means also that the Smart Badge application uses more than one hardware device. The devices used in this project are listed below but another configuration could also suit the hardware requirements (see Subsection 3.2.1 for an installation guide).

- UltraSparc 450 Server, which runs the Event Manager.
- Medio L200 RFID Reader, from Tagsys<sup>2</sup>. It provides four 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.
- HP WorkStation xw6200, to handle the Smart Server and the Smart Web Server

## 3.2 User Guide

### 3.2.1 Installation and Tutorials

#### Setting up the Sensing Layer

Technically speaking, the Sensing Layer is composed of two elements:

- The RFID hardware consisting of 1.. $n$  readers. In this case a Medio L200 sensor from Tagsys<sup>3</sup>. It is worth noting at this point that using the RFID simulator provided with the RFIDLocator, no physical reader is required to test the Smart Badge infrastructure, see 5.2.5 in [Gui05] for more information.
- The Event Manager: in this case the Sun Java System RFID Software Event Manager (v1.0 or above).

Installing these components is not always trivial and may require engineering skills. Despite of this fact, the installation procedure will not be detailed here as it was already extensively described in [Gui05] (Section 7.2).

#### Setting up the Environment Solving Layer

The central component of the Environment Solving Layer is the RFIDLocator. Furthermore, this J2EE EJB application needs to be deployed on a J2EE compliant Application Server. Any concrete Application Server could be chosen but the RFIDLocator best suits the Sun Java System Application Server (8.1 or above) together with Point Base RDBMS.

Again, the installation of the RFIDLocator will not be described here as it is the matter of Section 7.2 in [Gui05]. Note that one should install the version of the RFIDLocator located on the CD-ROM of the Smart Badge project and not the official version (version 1.0). Indeed, the former contains the additional Solvers programmed for the Smart Server. If you wish to install the version 1.0 anyway you will need to include the `SmartBadgeConcreteSolverSessionBean` and `SmartBadgeSingleTypeConcreteSolverSessionBean` classes prior deployment.

The configuration of the RFIDLocator in order for it to fit the needs of the Smart Badge infrastructure will be described in the paragraphs below.

---

<sup>2</sup>Tagsys is a french company expert in Auto-Id hardware

<sup>3</sup>The Smart Badge infrastructure was successfully tested with a Medio L100 RFID reader as well.

**Configuring the JMS Queue:** Firstly the common JMS queue needs to be setup. As mentioned before, this queue act as an interface between the RFIDLocator and the Smart Server. The RFIDLocator posts Service Requests on it while the Smart Server listens to the queue for incoming messages (requests). The procedure for creating this queue is similar to the one for the JMS components of the RFIDLocator (as described in 7.2.4 of [Gui05]):

The common JMS queue needs to be configured on the Application Server. We will briefly describe the required steps for the Sun Java System Application Server (8.1):

1. Start the Application Server (if not already started).
2. Open a new browser window and go to: `http://localhost:4848`. The system 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)DestinationResources from the left menu.
4. Click on “new”.
5. Enter `jms/RFIDLocatorOutputQueue` for the JNDI Name and select `javax.jms.Queue` for the type.
6. For the Value of the Name property enter `RFIDLocatorOPQ`. This is the name of the actual queue abstracted by the `jms/RFIDLocatorOutputQueue`.

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 `RFIDLocatorOPQ` for the Physical Destination Name and select `Queue` for the type.

The JMS elements needed for the communication between the RFIDLocator and the Smart Server are now set.

**Configuring the Environment:** Now that the RFIDLocator is setup we need to inform it about the environment it should monitor. This is done by parsing an XML file written accordingly to the Readers Configuration Formalism as described in Section 5.7 of [Gui05]. Basically such a file contains the location of the sensor within the physical (closed) environment as exposed in Subsection 2.2.2.

In order to get a better idea of this configuration, Figure 3.1 provides a sample XML file. The syntax and semantics of this fragment are extensively described in Section 5.7 of [Gui05]. However, some Smart Badge related information about it can be useful.

Firstly, the `BusinessLocationNumber` field corresponds to the unique identifier of a service point (see Subsection 2.2.3). For instance the first `BusinessLocationNumber` on Figure 3.1 corresponds to the `UNIFR_ENTRY` service point.

Secondly, the `JNDIENCName` field corresponds to algorithm we want to start when a user passes by this service point. As said in Subsection 2.2.2 two solvers were designed for the Smart Badge infrastructure:

---

```

<?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 L200">
    <LogicalAntenna>
      <Solver>
        <JNDIENCName>
         .ejb/SmartBadgeConcreteSolver
        </JNDIENCName>
        <MaxReadingTime>5000</MaxReadingTime>
      </Solver>
      <Location>
        <BusinessLocationNumber>
          UNIFR_ENTRY
        </BusinessLocationNumber>
        <Description>Entry of the UNIFR
        </Description>
      </Location>
      <PhysicalAntenna id="urn:epc:id:gid:1.1.1">
        <Action>IN</Action>
      </PhysicalAntenna>
      <PhysicalAntenna id="urn:epc:id:gid:1.1.4">
        <Action>OUT</Action>
      </PhysicalAntenna>
    </LogicalAntenna>
    <LogicalAntenna>
      <Solver>
        <JNDIENCName>
         .ejb/SmartBadgeSingleTypeConcreteSolver
        </JNDIENCName>
        <MaxReadingTime>0</MaxReadingTime>
      </Solver>
      <Location>
        <BusinessLocationNumber>
          AGEF_OFFICE
        </BusinessLocationNumber>
        <Description>Office of the AGEF
        </Description>
      </Location>
      <PhysicalAntenna id="urn:epc:id:gid:1.1.3">
        <Action>IN</Action>
      </PhysicalAntenna>
    </LogicalAntenna>
  </Reader>
</RFIDLocator>

```

---

Figure 3.1: Configuration of the physical environment

The screenshot shows the RFID Locator web application interface. At the top, there is a navigation bar with a logo, a welcome message, and links for 'logout', 'API, Documentation, etc.', and 'about...'. Below this is a menu on the left side with options like 'Create a new user', 'Configure the environment', 'Find a registered object', 'Attach or detach an RFID tag to an object', and 'Simulate PML events'. The main content area is titled 'Attach / Detach a Tag to a Business Object' and contains a form for 'Business Object and RFID Tag information'. The form includes fields for 'BusinessNumber:' (student\_irene\_blue), 'EPC URI (or tag id):' (urn:epc:id:gid:0.0.1), and 'SecurityLevel:' (confidential). There are radio buttons for 'Detach' and 'Attach', with 'Attach' selected. A 'Submit' button is at the bottom. The footer contains logos for Software Engineering Group, Sun microsystems, Universitas Padjadjaran, and Faculty of Science, along with a copyright notice for Dominique Guinard.

Figure 3.2: Adding a new user to be traced

- The first one `ejb/SmartBadgeConcreteSolver` (class name: `SmartBadgeConcreteSolverSessionBean`) is to be configured for service points requiring to detect the direction of the motion (e.g. entering the University or exiting it).
- The second, `ejb/SmartBadgeSingleTypeConcreteSolver` (class name: `SmartBadgeSingleTypeConcreteSolverSessionBean`) is used for the service points that do not need to capture the direction of the motion but only to notify the passing by of an object.

These solvers are based on a motion capture heuristic designed for the RFIDLocator and they both output Service Requests in the formalism developed for the Smart Server as described in 2.2.2.

To eventually parse the XML file, one may feel comfortable to use the provided web frontend functionality. The access to this interface is described in Subsection 7.1.2: “Access to the Main Functionalities” of [Gui05].

**Users Configuration:** The last thing to do is to inform the RFIDLocator about the users we want to monitor. Within the RFID locator, the assets one wants to trace are identified as `BusinessObjects`. Thus, to trace a user we need to register her as a new `BusinessObject`. This can be done through the Attach / Detach a Tag to a Business Object functionality of the API. This procedure is the matter of Subsection 7.1.2. in [Gui05] but a simple example is provided below:

If you want user Irene Blue to be traced by the RFIDLocator you need to fill the fields of the web frontend as pictured on Figure 3.2 With `student_irene_blue` being the unique identifier of Irene within the institution running Smart Badge and `urn:epc:id:gid:0.0.1` being the unique identifier contained in the badge she wears.

Once this is done Irene will be traceable within the predefined area as long as she carries her RFID badge with her.

### Setting up the Service Delivery Sub-Layer

As mentioned before, the Smart Server is a bundled J2EE application. Thus, to install it one needs to deploy it on a J2EE compliant Application Server configured with a MySQL Connection Pool.

**Creating a MYSQL Pool:** Prior deploying it on the target Application Server may need to be bound to MySQL. This procedure will be described for the Sun Java System Application Server 8.1:

1. Start the Application Server (if not already started).
2. Open a new browser window and go to: <http://localhost:4848>. The system 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>JDBC>ConnectionPool from the left menu.
4. Click on “new”.
5. Enter MySQLPool as Name, javax.sql.ConnectionPoolDataSource as Resource Type, and select mysql as Database Vendor.
6. Click next and enter com.mysql.jdbc.jdbc2.optional.MysqlDataSource as Datasource Classname.
7. Click next and enter jdbc:mysql://localhost/smartbadge as URL, root (or your DB user) as user and your DB password for the password field.
8. Now click finish.

The last step creates a new Connection Pool. We now bind a JDBC resource used by the Smart Server to the newly created pool:

1. Select Resources>JDBC>JDBCResources from the left menu.
2. Click on “new”.
3. Enter jdbc/mysql as Name and select MySQLPool as Pool Name.

This creates the JDBC resource for our application. We can now deploy the Smart Server.

**Deploying the Smart Server:** There exist various ways of deploying a J2EE application on a server. To deploy it on the Sun Java System Application Server achieve the following steps:

1. Start the Application Server (if not already started).
2. Open a new browser window and go to: <http://localhost:4848>. The system 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 EnterpriseApplications>Deploy browse for the smartserver.ear archive contained in the SmartServer/dist directory of the CD-ROM and choose next.

4. After a few seconds the Smart Server should be up and deployed on your Application Server.

After these steps the Sensing layer, Environment-solving layer and the Service Delivery sub-layer should be setup. The only remaining part is the Service Management layer which installation is described below.

### Setting up the Service Management Sub-Layer

Using PHP technology, the Service Management Sub-Layer needs a properly configured server to process PHP scripts before they are submitted to the web browser. To access correctly to the web interface, the Apache<sup>4</sup> server and a MySQL database server must be installed prior to continue. Once these goals achieved, follow the steps below:

1. Copy the Smart Web Server source files located in the Applications/SmartWebServer directory of the project CD-ROM to the directory of your choice (called [SWSPath] in the next paragraphs).
2. Configure the Apache server by modifying his configuration file (generally “apache2.conf”):

- Enable lecture of the custom .htaccess permission file:

```
<Directory [SWSPath]\>
    AllowOverride All
<\Directory>
```

- Set default charset and turn on the PHP module:

```
AddDefaultCharset ISO-8859-1
AddType application/x-httpd-php .php
AddType application/x-httpd-php-source .phps
```

- Enable URL-ReWriting from console (as URL used has been standardized and needs transformation):

```
$sudo a2enmod rewrite
```

3. Initialize the database (using a database administration tool like PHPMyAdmin<sup>5</sup>):

- Create an user called “smartbadge” with password “smartbadge” owning all privileges.
- Create a database called “smartbadge”.
- Use the SQL script located at Applications/SmartWebServer/sql directory to create tables and insert all required data (like web site configuration or language variables).

From now, the Smart Web Server should be accessible from a web browser by entering <http://localhost/smartbadge>.

---

<sup>4</sup><http://www.apache.org>

<sup>5</sup><http://www.phpmyadmin.net>

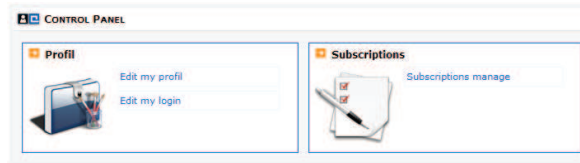


Figure 3.3: Features available from the user control panel

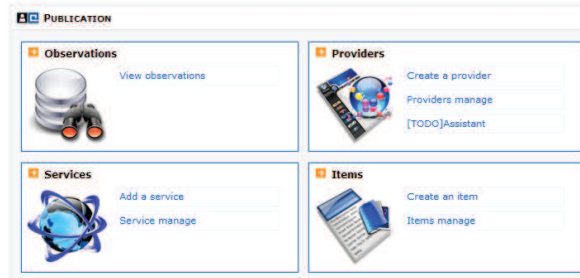


Figure 3.4: Features available from the editor control panel

### Using the Smart Web Interface

Once the Smart Web Server has been properly installed you should now be able to create a new account (Connection)CreateAccount) or log in directly with one of the predefined accounts:

- As *administrator* (access to all features):  
username: **admin**  
password: **admin**
- As *editor* (access to publishing features):  
username: **editor**  
password: **editor**
- As *user* (access to subscription features):  
username: **user**  
password: **user**

Now, if you wish to setup a contextual service containing items to send to registered users, you can follow the tutorial below (please use an administrator account to access all features). Remember that users are automatically registered to services that you create, so you do not need to associate them with services. If one user don't wants to receive informations about a service, he has the possibility



Figure 3.5: Features available from the admin control panel

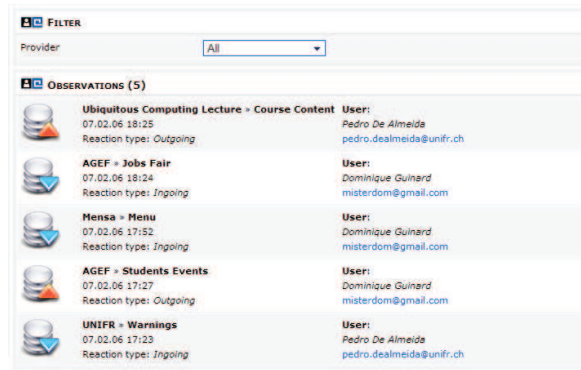


Figure 3.6: Listing of smart observations

to unsubscribe using the subscription interface (Options)DisplayMenu)Subscriptions)Subscriptionsmanage).

1. Create a new service point (Options)DisplayMenu)ServicePoints)CreateaServicePoint) representing an area offering services. Note that the “Identifier” must be the same as the one provided into the Smart Server.
2. Create a new communication support (Options)DisplayMenu)Communicationsupports)AddaNewSupport) to specify the way the data will be sent. Take in account that the “Driver” name must match the class implemented in the Smart Server.
3. Create a new provider (Options)DisplayMenu)Providers)CreateaProvider) to handle a set of services.
4. Create a new service (Options)DisplayMenu)Services)AddaService) dependent from the provider created on step 3, using the service point created on step 1 and transmitting data through the communication support created on step 2.
5. Add items (Options)DisplayMenu)Items)CreateanItem) to the service created on step 4 and contextual informations are ready to be sent to registered users.
6. Check if observation have been done using the observation interface (Options)Observations or (Options)DisplayMenu)Observations)ViewObservations).

# 4

## Evaluation

---

<b>4.1 Adherence with the Specification . . . . .</b>	<b>25</b>
<b>4.2 Experiments . . . . .</b>	<b>25</b>
<b>4.3 Analysis of the Results . . . . .</b>	<b>26</b>
4.3.1 Impressions . . . . .	26
4.3.2 Technical Limitations . . . . .	27

---

### 4.1 Adherence with the Specification

A significant part of our work was dedicated to specification. Thus, after a substantial amount of hours spent on this project we came out with a quite precise and documented (mainly using UML and Entity-Relationship diagrams) specification. This was time consuming but worth it as our final application respects the specification. The only point we did not fullfill is the handling of formalized SMS messages by a mobile application. Rather than to a lack of knowledge, this failure is due to the fact that one member of our team decided not to attend the lecture anymore...

### 4.2 Experiments

Three real tests were conducted on the Smart Badge Infrastructure. Each experiment was done in the RFID Laboratory at the University of Fribourg (Switzerland)<sup>1</sup>. For this purpose, the lab was transformed into the UGS (“Uni Goes Smart”) room for a couple of weeks.

The basic idea behind this use case is to imagine a number of services that could augment the user interactions with the building of our University. For this purpose we transformed the lab into a Kinesthetic User Interface by setting up the services summarized on Figure 4.1.

1. The first experiment was done by an external “guinea-pig” (non-computer scientist): Carole Betti.
2. The second was run by two team members of the Smart Badge project: Pedro Dealmeida & Dominique Guinard.
3. Finally, the third was done by the interested fellows of the Ubiquitous Computing Lecture.

Provider	Trigger	Support and Service
UNIFR	On entering the University's main building (IN)	SMS: Reminder for exam registration, EMAIL: Welcome message.
UNIFR	On exiting the University (OUT)	EMAIL: Summary of the conferences to come.
AGEF	On standing by AGEF's office (IN)	SMS: upcoming student's parties, EMAIL: latest job opportunities.
Ubiquitous Computing Lecture	On sitting at a desk of the lecture room (IN)	EMAIL: summary of today's lecture, WEB-FRONTEND: attendees (for the professor).

Figure 4.1: Services settings for the UGS use case

During our guinea-pigs were asked to:

- Filling their Smart Badge accounts through the web-interface.
- Wear a Smart Badge (pasted onto their shirts).
- Enter the lab, standing for the University (triggering the service 1 on Figure 4.1).
- Stand in front of AGEF's office while reading some news placarded on the wall (triggering service 3 on Figure 4.1).
- Sit at a desk and study a paper about Agents in the room of the Ubiquitous Computing lecture (triggering service 4 on Figure 4.1).
- Exit the lab, standing for the University (triggering service 2 on Figure 4.1).
- Check their emails and cellular-phones in order to see if the services were delivered.
- Provide a feedback about the application.

In total, each subject spent about twenty minutes in the lab.

## 4.3 Analysis of the Results

### 4.3.1 Impressions

First of all, the infrastructure made a strong impression on the subjects. They liked the approach of interacting with the system simply by “being”. That is: without actually having to use an interface (or through what we call a Kinesthetic User Interface).

In the mean time, this (sometimes) unconscious way of interacting with the system raised the same question by our testers: “Isn't the application sort of tracking us?”. At this point in the report it should be clear to anyone that the infrastructure **is tracking the user**. Indeed, it is the essence of our interface of a new kind.

<sup>1</sup><http://diuf.unifr.ch/>

We are conscious of the fact that it may raise some important privacy and security issues. To avoid abuses (i.e. using the system against its users) clear policies should be introduced by the institution providing the Smart Badge infrastructure. Additionally, some testers made relevant remarks about the influence of the electromagnetic field generated by the readers. This, issue is of great importance when you think of the Smart Badge infrastructure installed in a “every-day” environment. However, discussing this fact is beyond the scope of this report, mostly because it requires knowledges we do not have and because it is a contentious and still disputed subject.

### 4.3.2 Technical Limitations

Another important fact that all our subjects noticed is the limitations due to the technology. Indeed, the RFID readers we were using sometimes failed to capture the transponders. As a result some services were never delivered.

Even if these limitations fairly exists three points about them are important to note:

- The RFID hardware we are using is not of the best kind. The Medio Readers have average performances for a broad range of purposes (which makes them very handy in a laboratory) but are not the best readers for our use-cases.
- We had no time to “tune” the RFID hardware and the smart room we built. This, we believe, would have improved the efficiency of the readers.
- The RFID technology is evolving really fast. This fact has two consequences:
  - The (still quite high) price for performing and adapted readers should lower significantly in a near future.
  - The average range of reading as well as the accuracy of the readers is constantly improving.

# 5

## Future Works

### 5.1 Possible Extensions

Even if the Smart Server and the Smart Web Server are functional, many extensions and improvements can still be made. The most important from our view-point are listed below:

- Create a mobile application (e.g using J2ME) for aggregating and managing the services delivered to handhelds.
- Simplify the service points configuration as well as the user management. Indeed, in the current state some required<sup>1</sup> information between the Environment-Solving layer and the Service layer are redundant, which does not ease the task of the administrator.
- Simplify the creation of new elements (services, service points, etc.) in the Smart Web Server. This could be achieved for instance by creating “Wizards”.
- Simplify the installation of the infrastructure. Indeed, as the Smart Badge project is a system rather than a single application its installation is not always straightforward to the beginner.
- Create a broader range of concrete Transmitters in order to permit more use cases.
- Further contextualize the provided services<sup>2</sup>. For instance by matchin user preferences (for food, leisure, timing, etc.) to services.
- Allow the creation and delivery of dynamic services. I.e. services which content is evolving (automatically) overtime.
- etc.

---

<sup>1</sup>The users and the service points have to be configured in both the Environment-Solving layer and the Service Management sub-layer.

<sup>2</sup>See [HIR02] for a good approach on how to model context.

# 6

## Project Management

---

6.1	Team Composition . . . . .	29
6.2	Individual contributions . . . . .	30

---

### 6.1 Team Composition

The Smart Badge projet has been developed by Dominique Guinard and Pedro De Almeida, both students at the University of Fribourg, under the supervision of Pr. Dr. Béat Hirsbrunner and his assistant Dr. Vincenzo Pallotta. This projet was proposed by the “Ubiquitous Computing: Coordination and Fundamental Tools” course and achieved during the winter semester 2005/2006.

## 6.2 Individual contributions

Tasks	Dominique	Pedro
<ul style="list-style-type: none"> <li>• Environment Installation               <ul style="list-style-type: none"> <li>- Operating System (Linux Ubuntu 64 bits)</li> <li>- Sun Application Server</li> <li>- Pointbase database server</li> <li>- Apache web server</li> <li>- MySQL database server</li> <li>- PHPMyAdmin</li> <li>- VSFTP server</li> </ul> </li> </ul>	50%	50%
<ul style="list-style-type: none"> <li>• Sensing Layer               <ul style="list-style-type: none"> <li>- Configuring the Event Manager</li> <li>- Setting up the RFID Hardware</li> </ul> </li> </ul>	90%	10%
<ul style="list-style-type: none"> <li>• Environment-Sensing Layer               <ul style="list-style-type: none"> <li>- Installing/Configuring the RFIDLocator</li> <li>- Writing new the new solvers</li> <li>- Configuration of the inter-layer communication interfaces (JMS queues)</li> </ul> </li> </ul>	90%	10%
<ul style="list-style-type: none"> <li>• Service Delivery Layer               <ul style="list-style-type: none"> <li>- Entity-Relationship database model</li> <li>- Entity Beans</li> <li>- Service Processor</li> <li>- Service Transmitter</li> <li>- Concrete Transmitters</li> <li>- Message Driven Beans</li> <li>- Deployment descriptors</li> <li>- Design of the Transmitters</li> <li>- Configuration of the inter-layer communication interfaces (JMS queues + Transacted database)</li> <li>- Configuration of the SMTP communication</li> </ul> </li> </ul>	80%	20%
<ul style="list-style-type: none"> <li>• Service Management Layer               <ul style="list-style-type: none"> <li>- Entity-Relationship database model</li> <li>- Database structure</li> <li>- Web Interface                   <ul style="list-style-type: none"> <li>User Management</li> <li>Service Points Management</li> <li>Communication Supports Management</li> <li>Providers Management</li> <li>Services Management</li> <li>Items Management</li> <li>Observations</li> </ul> </li> </ul> </li> </ul>	20%	80%
<ul style="list-style-type: none"> <li>• Installation of the Uni-Goes-Smart use case               <ul style="list-style-type: none"> <li>- Reader's configuration</li> <li>- Service configuration</li> <li>- Environment configuration</li> </ul> </li> </ul>	50%	50%
<ul style="list-style-type: none"> <li>• Documentation</li> </ul>	50%	50%

# Bibliography

- [FGL06] Patrik Fuhrer, Dominique Guinard, and Olivier Liechti. RFID: From Concepts to Concrete Implementation. In *Proceedings of the International Conference on Advances in the Internet, Processing, Systems and Interdisciplinary Research, IPSI - 2006 Marbella, February 10-13, 2006, Marbella, Spain*. IPSI Belgrade, Serbia, Serbia and Montenegro, February 2006. [Retrieved February 16, 2006, from <http://diuf.unifr.ch/people/fuhrer/publications/external/RFIDIPSI.pdf>].
- [Gui05] Dominique Guinard. Bachelor Work: Radio Frequency Identification: Evaluation of the Technology Supporting the Development of an Assets Tracking Application, 2005. [Retrieved February 27, 2006, from <http://www.gmipsoft.com/rfid>].
- [HIR02] Karen Henricksen, Jadwiga Indulska, and Andry Rakotonirainy. Modeling context information in pervasive computing systems. In *Pervasive 2002, LNCS 2414*, pages 167–180. Springer-Verlag Berlin, 2002.
- [HMSS03] Uwe Hansmann, Lothar Merk, Martin S., and Nicklous Thomas Stober. *Pervasive Computing*. Springer, second edition, 2003.
- [HSaD<sup>+</sup>05] Michael N. Huhns, Munindar P. Singh, Mark Burstein and Keith Decker, Edmund Durfee and Tim Finin, Les Gasser, Hrishikesh Goradia, Nick Jennings, Kiran Lakkaraju, Hideyuki Nakashima, H. Van Dyke Parunak, Jeffrey S. Rosenschein, Alicia Ruvinsky, Gita Sukthankar, Samarth Swarup, Katia Sycara, Milind Tambe, Tom Wagner, , and Laura Zavala. Research directions for service-oriented multiagent systems. *IEEE Internet Computing*, 2005. [Retrieved February 5, 2006, from <http://eprints.ecs.soton.ac.uk/11736/01/ic05.pdf>].
- [IBM05] IBM. IBM Pervasive Computing. Website, 2005. [Retrieved February 21, 2006, from <http://ibm.com/pvc>].
- [Lie05] Olivier Liechti. Rfid: Middleware and integration with the information system. Conference, December 2005. [Retrieved February 5, 2006, from [http://diuf.unifr.ch/softeng/rfid/download/rfid\\_fribourg\\_liechti.pdf](http://diuf.unifr.ch/softeng/rfid/download/rfid_fribourg_liechti.pdf)].
- [Mic05] Microsoft. Microsoft research. Website, 2005. [Retrieved February 21, 2006, from <http://research.microsoft.com/>].
- [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.

- [Wei88] Mark Weiser. The computer for the 21st century. *Scientific American*, 1988. [Retrieved February 21, 2006, from [http://www.media.mit.edu/resenv/classes/MAS961/readings/weiser\\_reprint.pdf](http://www.media.mit.edu/resenv/classes/MAS961/readings/weiser_reprint.pdf)].

# Index

- Apache, 22
- API, 12
- Better Sensing Technologies, 1
- Business Object, 20
- Cellular phone, 6
- Deployment, 21
- Electronic Product Code, 9
- Entities, 14
- Entity-Relationship Model, 14
- EPC, 9
- Evaluation, 25
- Event Manager, 10, 17
  - adapters, 10
  - filters, 10
  - loggers, 10
- Extensions, 28
- Functional Description, 8
- Future Works, 28
- Handheld, 5
- IBM, 5
- Installation, 17, 21, 22
- Invisible Computer, 1
- J2ME, 28
- Javadoc, 12
- JMS, 11
- Kinetic User Interface, 4, 7
- KUI, 4, 7
- Locator, 11
- Medio
  - L100, 17
  - L200, 17
- Microsoft, 5
- MySQL, 16
- Operational Instructions, 16
- PHP, 16
- Physical Markup Language, 9
- PML, 9
- Project Management, 29
- RFID, 5, 11
- RFIDLocator, 11, 17
- Savant, 10
- Sensor, 11
- Service Delivery and Resolution, 7
- Service Management Layer, 14, 22
- Smart Badge Service Requests, 12
- Smart Home, 5
- Smart Server, 21
- Smart Web Interface, 23
- Smart Web Server, 14, 23
- SMS, 6
- Software, 16
- Sun Java System RFID Software, 17
- System Requirements, 16
- Tag, 11
- Tagsys, 17
- Tutorials, 17
- URL-ReWriting, 22
- Use, 23
- Use Cases, 6
- User Guide, 17
- User Roles, 14
- Weiser, Mark, 1
- XML, 12