OW2 AspireRFID
RFID Suite branch

Global design

Document	Global conception
Version	0.2
Last modification	05/06/07
Status	Final
Producer	Université Joseph Fourier (Grenoble 1), LIG/ADELE
Authors	FORNACIARI François SURREL Guillaume VAUDAUX-RUTH Guillaume
Number of pages	14
Author	VAUDAUX-RUTH Guillaume
Validated by	DONSEZ Didier

Table of revisions

Version	Date	Modifications
0.1	21/05/07	Beginning of document
0.2	05/06/07	Addition of schemas

Table of contents

1 Introduction

1.1 Goal of this document

This document presents the global conception. It defines the architecture of the RFID Suite project and presents goals of each component and dependencies between them.

1.2 Document range

This document is written for :

- Project leader : Didier Donsez
- Developer team :
  - François Fornaciari
  - Guillaume Surrel
  - Guillaume Vaudaux-Ruth

1.3 Glossary

See the external file "glossaire.pdf" (only available in French).

2 Architectural view

The software architecture is almost given by the client, same case for the technologies (see “Software requirements specifications for RFID Suite” for more details)

2.1 General architecture

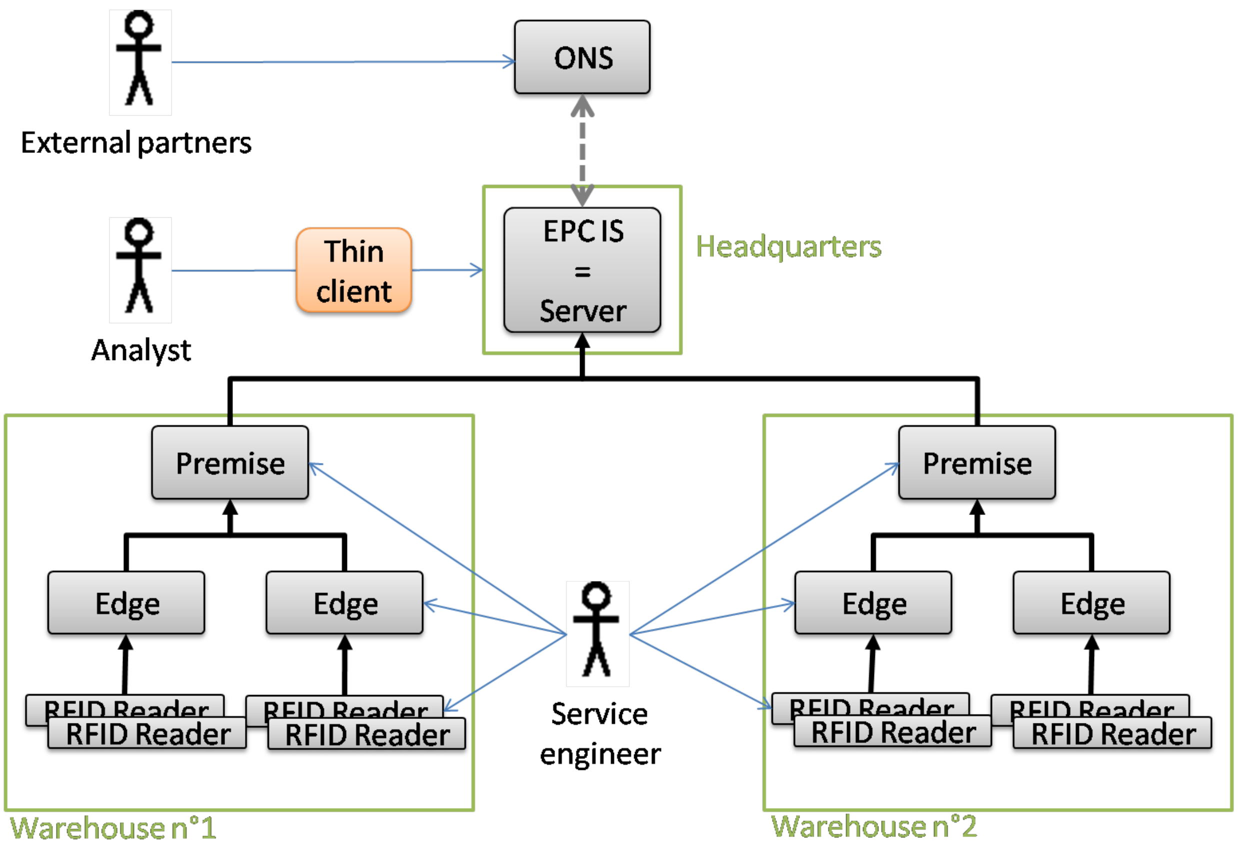

Figure 1: Global architecture with stakeholders

2.2 Reasons for this architecture

It becomes clearer and clearer that deploying such services in centralized architectures where one server collects directly sensor data is no longer a reasonable solution. Limitations are encountered when dealing with an important number of sensors and their measurements. Today, new architectures appear for the equipment's remote exploitation: the use of standardized and smart Internet gateways between sensors and IT servers. The edge deals with equipment administration and the collected data. Furthermore, a mediation protocol can be used between edges and servers, alleviating the communications, filtering and aggregating information during data transport (figure 2).

The most important demand for sensor-based applications middleware is to ensure support for communication. Communication in sensor-based applications is somehow special. We state that classical synchronous client-server communication is not suited for such application. In the meantime, data-centric communication that is based on a paradigm more similar to content-based messaging is more adapted in this context. Moreover, event-based messaging matches much better the requirements of sensor-based applications than traditional request-reply schemes.

2.3 Communication modules

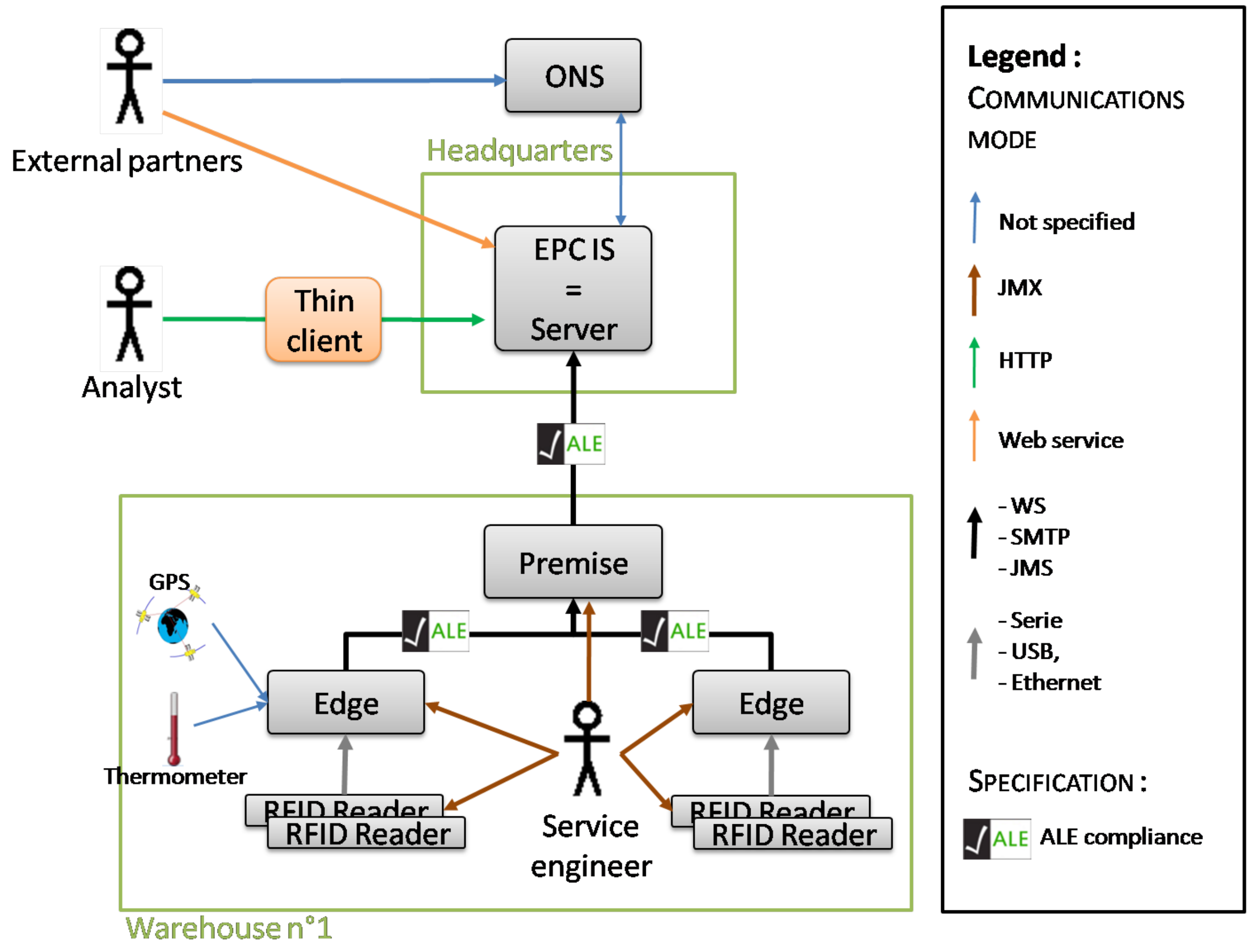

Figure 2: Communication module

2.4 Communication interfaces

In the chosen architecture, we have two interface types :

- Internal event : an event between two components on the same platform (on the edge for example). In this case, we use the EventAdmin (OSGi events). This event includes user attributes defined in a map.
- Event published through the network. The same event type is used but a mechanism called the "dispatcher" creates and sends the message using the protocol defined in the event attributes.

3 Module view

3.1 *Edge*

3.1.1 General Description

The Edge is composed by a set of OSGi bundles. It allows to aggregate and filter the information sent by RFID Readers. Aggregating and filtering part is defined by the ALE specification. Other data like GPS localization or temperature is also read by the Edge. The data handled by the edge is placed into reports which are sent to the premise or directly to the server. The information can be transmitted through one or few types of communication.

3.1.2 Constraints

Edges must run over Apache Felix and Equinox, which are two open source OSGi™ implementations.

3.1.3 Technical description / Solution

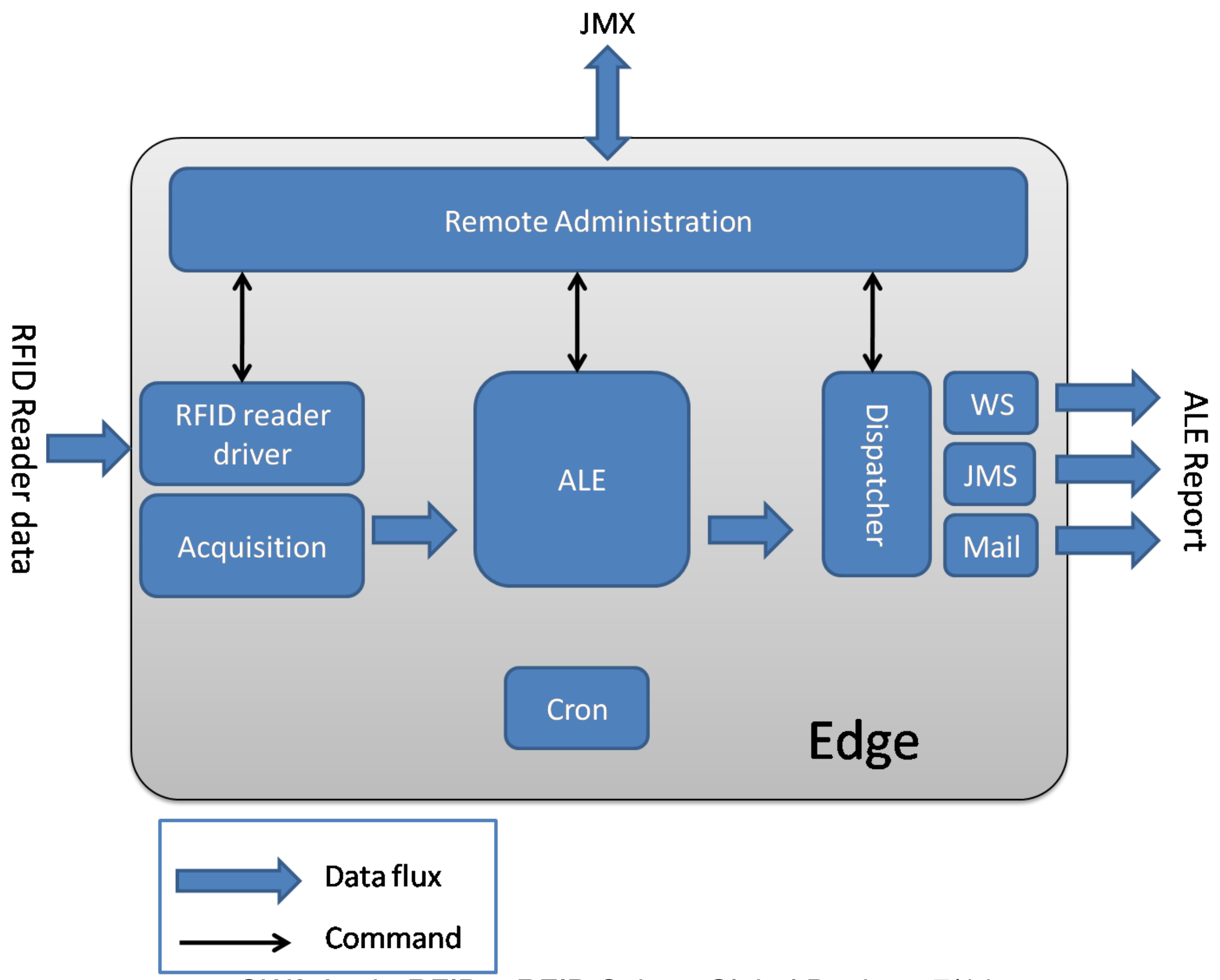

Figure 3: Edge architecture

Acquisition : receive data and convert them to a more usable data. This module manage the RFID readers to interrogate the physical RFID reader device connected to the edge.

The information read from captors will be filtered in the ALE module. To limit the dependencies between reader and ALE module, information is propagated through events: all installed reader interfaces generate OSGi events, and the filter listen to these events. In this way, the ALE module do not need to know the type and the number of installed reader interface.

The acquisition module periodically asks to the reader the list of present RFID tags, and generate an event to propagate this information to filter module. The read period is not specified by the ALE interface. It may depend on the reader capability, on its place (read static information in stock, or detect moving tags through gate...). This period must be modifiable by configuration.

The readers protocol is not standardized, and we must have a different module for each known reader type. For each reader type, we may have one or more readers connected to the edge. To be able to configure read periods independently on each reader, an instance of reader driver is deployed for each installed reader.

ALE : this bundle implements the ALE specification version 1.0. ALE provides a standardized method to aggregate and filter data from readers. ALE defines also an XML schema (that can be extended) to allow standardized exchanges between many modules.

If more than one business application needs to collect RFID information, the needed filters may be different for each application. In this case, we have two solutions:

- create a filter instance for each application. Each filter sends its report to the corresponding application. In this case, the same information may be duplicated, and travel several times through the network.
- The edge filter is the sum of the needed filters, all needed information is collected, and stored to the database, and each application uses again its filter to interrogate the database.

To be flexible on each solution, filters are generated, and configured to get either reader information, or other filter information in input. In the same way, the filter generates an ALE XML report, of an object structure which may be an input of another filter.

Dispatcher : collected information needs to be sent to the premise or the server. This module dispatches messages through the protocol defined in them.

Remote administration : this bundle is a JMX Agent. It allows to register MBean on the JMX server which is embedded to it. Any bundle can “expose” a JMX service, which will be automatically detected and registered by the JMX Agent. The remote administration allows to configure readers, to define filters and to manage the dispatcher.

Cron : it is a bundle that provides an OSGi™ service to execute a task periodically. For example, it is used to send reports automatically.

RFID reader driver : bundle that corresponds to a driver for a RFID reader.

3.2 Premise

3.2.1 General description

The premise is an intermediary between many edges and one server . It has two main goals. First, it is used to receive reports which come from its attached edges. Moreover, these reports have to be aggregated and filtered in the same manner that for the edge. Second, the premise sends reports or alerts to the server. The information can be received and transmitted through one or few types of communication.

3.2.2 Constraints

Premises must run over Felix and Equinox, which are two open source OSGi™ implementations.

3.2.3 Technical description / Solution

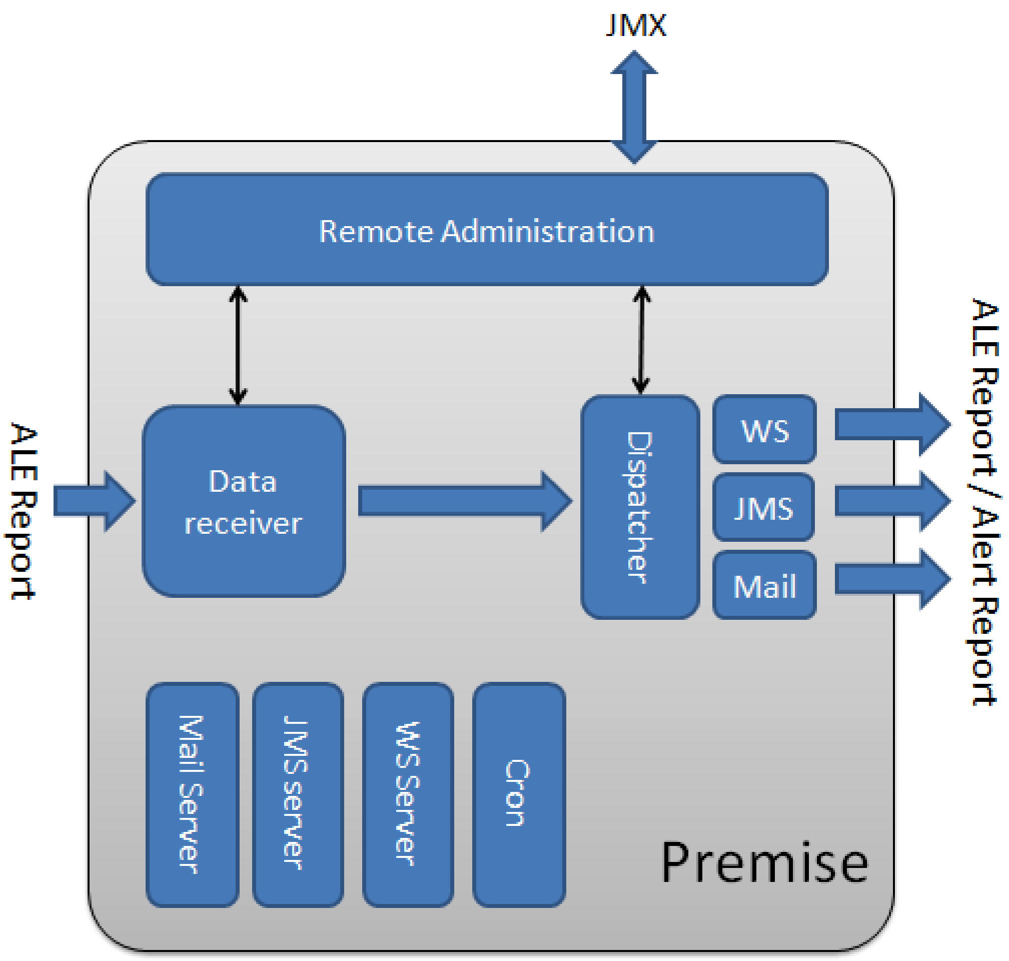

Figure 4: Premise architecture

Mail Server : it is the Java Email Server (JES) packaged in a bundle. As this server runs on OSGi, it provides a service to get e-mail messages. This service will be used by the data receiver to read reports.

JMS Server : it is a Java implementation of JMS called Joram packaged in a bundle. This bundle receives messages from specified topics that will be treated by the data receiver.

WS Server : it is a Java SOAP framework called XFire packaged in a bundle. Like the previous two bundles, this server provides web services that can be called by the edges. On each remote call of a service, the report included in the message is treat by the data receiver.

Data receiver :
To generalise the reception of messages, the data receiver contains a dispatcher. Each message reception triggers the call of the dispatcher which reads the report and make specific processes. The The dispatcher implements three main functions :

- The report received contains current report sets : all members contained in the report are transformed in OSGi events to be filtered with ALE. Thus, the edge play the role of reader for the premise.
- The report received contains additions or deletions report sets : the report is sent directly to the server.
- The premise generates alerts from read RFID tags and send it to the server.

We have seen that all reports received through three different communication ways are treat by the data receiver dispatcher. A same report may be sent using at least two communication protocols. Thus, the data receiver will receive two identical reports and have to ignore the doubloons. To realize the redundancy control, the message contains an unique identifier composed by a integer and the gateway name. For example, identical reports sent by a gateway through JMS and SMTP have the same identifier.

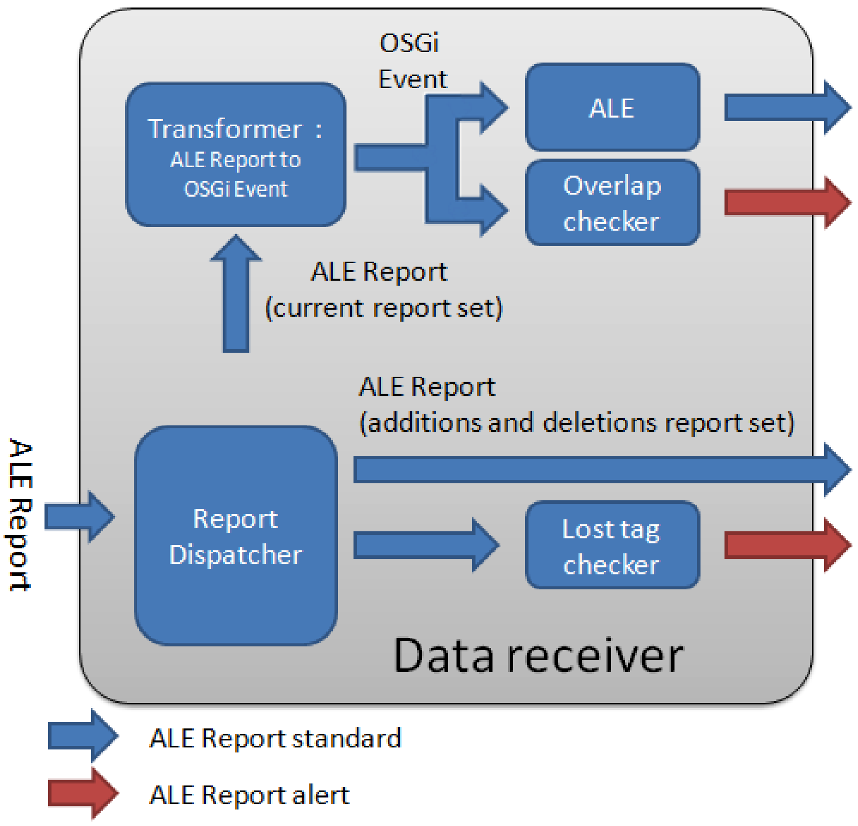

Figure 5: Data receiver architecture

The dispatcher :
This module is also used to send reports. Depending on the attributes of the message, the dispatcher has to call the specific protocol to transmit the report to the server. For each protocol type, a bundle is in charge of the sending and provides an OSGi service which can be requested by the dispatcher.

Remote administration :
As in the edge, the JMX administration is available thanks to the JMX Agent bundle. It allows to define filters, to configure the data receiver and to manage the dispatcher.

3.3 Server

3.3.1 General description

The server centralizes and stores all information. It receives reports from many premises, and maybe from many edges, what doesn't change anything in the manner to read data because the reception is transparent for the server. In fact, the data exchanged between the edges, the premises and the server respect the ALE specification.
The two other main goals of the server are to present the received information in a web interface and to provides web services for the partners which want to access to the collected data.

3.3.2 Constraints

Application server can be JOnAS or JBoss.

3.3.3 Solution

In order to be compliant with these two application servers, we define maven compilation profiles which execute specifics tasks depending on the server type. For example, some classes of the server are executed only on JOnAS and have to be excluded from the EAR deployed in JBoss.

The application performs four functions :

- The reports are stored in a database by using entity beans.
- The web interface is composed of JSP and session beans. Theses beans are commons for the two application servers.
- The server receives reports through asynchronous JMS messages, web services or session beans for the reception of mail messages. Theses beans are different for each application server.
- Some collected data is exposed through web services. Theses beans are different for each application server.

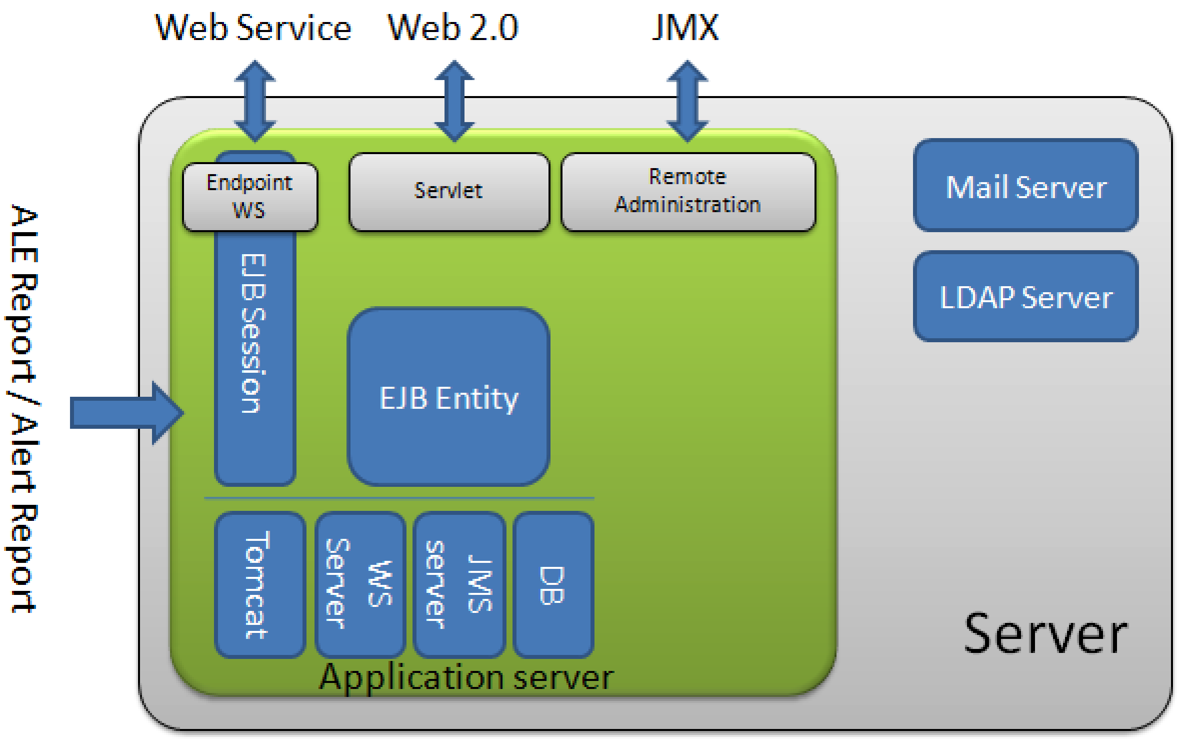

Figure 6: Server architecture

3.4 Object Name Service (ONS)

3.4.1 General description

An ONS provides a link between an EPC and an EPC IS. In fact, each company is in control of multiple blocks of EPCs and stores information about them. It's an important part of the EPC Global architecture which allows two companies (partners) to communicate together without the first company knows before the existence of the second. This server is comparable to the DNS for the Internet. Only few companies like VeriSign are in charge to provide this service.

3.4.2 Freedom

We only need to code an “ONS simulator”. Client don't compel us to respect ONS specification. The ONS needs to work only for a local demonstration.

3.4.3 Solution

The link between EPC IS and the sub-part of the EPC , the “EPC Manager Number”, is stored in the “Directory”. The “Query process” lookup in the Directory to answer requests. When an EPC IS needs information about an EPC, it makes a request to the ONS which returns it the address of the Web Service. The EPC IS that provides the service only permits authenticated companies to access to this service.

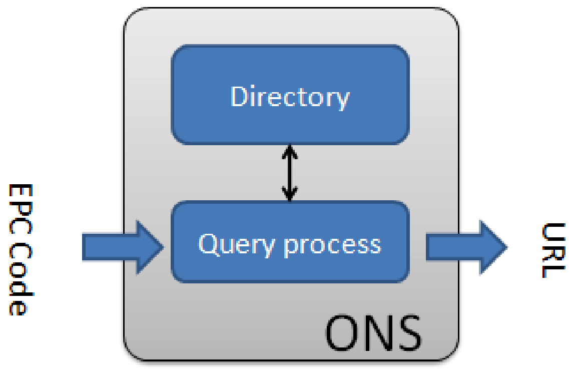

Figure 7: ONS architecture

4 ALE Extension

The ALE specification allows to add extensions to the XML schema. We use this extension to add the following information in the report schema:

- Reader identification in each member of ALE report : in ALE schema, the reader list is defined for the event cycle specification, but not defined for each member.
- Gateway address
- Reader GPS coordinates. On the edge, each reader stores its position. In our project, these coordinates are static, or configured by a JMX function. It will be possible to compute this parameter by a GPS sensor reader. This allows to have a moving reader, and to associate the current position to the RFID information.
- Reader temperature. On the edge, each reader collects the current temperature. In our project, this temperature is static, or configured by a JMX function. It will be possible to compute this parameter by a temperature sensor reader.
- Report set. An ALE event cycle report may include multiple reports of different types. Theses types are defined in the event cycle specification. In order to not always include the specifications in the report himself, we have decided to add a new attribute in the report XML node which represents the report set.

5 Appendix

5.1 Composition of an EPC

It contains the “EPC Manager” field which defines the unique owner identifier.

Element	Header	EPC Manager	Object Class	Serial Number
Bits	8	34	20	34