Collaborative Project

ASPIRE

Advanced Sensors and lightweight Programmable middleware for Innovative Rfid Enterprise applications

FP7 Contract: ICT-*215417*-CP

WP3 – RFID Middleware Infrastructure

Public report - Deliverable

Core ASPIRE Middleware Infrastructure (Final Version)

Due date of deliverable:	M30
Actual Submission date:	

Deliverable ID:	**WP3/D3.4b**
Deliverable Title:	Core ASPIRE Middleware Infrastructure (Final Version)
Responsible partner:	AIT
Contributors:	John Soldatos, Nikos Kefalakis, Nektarios Leontiadis, Nikolaos Konstantinou – AIT Nathalie Mitton, Loïc Schmidt, Roudy Dagher - INRIA Mathieu David, Bayu Anggorojati, Simone Frattasi, Neeli Prasad - AAU Didier Donsez, Gabriel Pedraza, Kiev Gama – UJF
Estimated Indicative Person Months:	12

Start Date of the Project:	1 January 2008	Duration:	36 Months

Revision:	1.9
Dissemination Level:	PU

Document Information

Document Name:	**Core ASPIRE Middleware Infrastructure (Final Version)**
Document ID:	**WP3/D3.4b**
Revision:	**1.9**
Revision Date:	**21 March 2011**
Author:	**AIT**
Security:	**PU**

Approvals

	Name	**Organization**	**Date**	**Visa**
Coordinator	Neeli Rashmi Prasad	CTIF-AAU		
Technical Coordinator	John Soldatos	AIT		
Quality Manager	Anne Bisgaard Pors	CTIF-AAU		

Reviewers

Name	**Organization**	**Date**	**Comments**	**Visa**
Mathieu David	CTID-AAU	18 Jun 09		
Didier Donsez	UJF	18 Jun 09		

Document history

Revision	**Date**	**Modification**	**Authors**
0.1	21 May 09	First draft	John Soldatos, Nikos Kefalakis, Nektarios Leontiadis
0.2	28 may 09	Section 4	Loïc Schmidt
0.3	29 may 09	Section 6.2	Nathalie Mitton
0.4	4 Jun 09	Updated connector (12.3), added context analysis (11)	Nektarios Leontiadis
0.5	12 Jun 09	Updates in sections 6, 7, 9, 13, 17, 18	Nektarios Leontiadis
0.6	12 Jun 09	Section 7	Nikos Kefalakis
0.7	15 Jun 09	Section 10	Mathieu David
0.8	16 Jun 09	Paragraphs 8.2, 8.3	Nikos Kefalakis
0.9	16 Jun 09	Section 3, 5.2, 5.3	Kiev Gama, Didier Donsez
1.0	17 Jun 09	Executive Summary, Introduction	Nektarios Leontiadis

1.1	18 Jun 09	Minor changes/corrections	Mathieu David
1.2	18 Jun 09	Minor changes/corrections	Didier Donsez
1.3	19 Jun 09	Final version	Nektarios Leontiadis
1.4	8 Apr 10	First Draft for version 4.3b, numerous corrections, updated section 4, new material in sections 6.2, 8.1, 8.2, Added list of acronyms and section 13	Nikolaos Konstantinou
1.5	22 Apr 10	Updated sections 5.3, 5.4, added new material in 7.2, 7.3, 7.4	Roudy Dagher, Loïc Schmidt
1.6	10 Jun 10	Added F&C OSGi implementation description and sensor data inclusion details	Gabriel Pedraza,
1.7	11 Jun 10	Gathered future steps in one section, wrote conclusions, updated and merged “current implementation status” sections, moved EPCIS capture interface in the EPCIS section, added Business Intelligence section	Nikolaos Konstantinou
1.8	21 Jun 10	Executive summary, Introduction	Nikolaos Konstantinou
1.9	24 Jun 10	Quality control	Bayu Anggorojati, Nikolaos Konstantinou

Content

Executive Summary

In this deliverable, we describe the developments related to the final version of the core ASPIRE middleware infrastructure. These components consist of both licensed and consortium-developed source code and are based on the ASPIRE architecture and the end user requirements that have been collected in the scope of Work Package 2.

Being fully in line with the ASPIRE middleware architecture, the design and implementation decisions for the ASPIRE middleware infrastructure have emphasized on the widest possible adoption of the RFID technology, on the extensive and extensible configurability of these components and the total compatibility with the ASPIRE IDE, which was developed in the scope of Work Package 4.

In principle, the ASPIRE middleware is able to operate with any reader platform regardless of vendor, frequency and supported functionality. Likewise, the ASPIRE middleware supports various tag formats and legacy IT information system languages. This freedom of choice is perfectly in line with both the “open” nature of the middleware and the requirements of the Small Medium Enterprises (SME’s). Avoiding vendor and technology lock-in is a major requirement from the SME community with respect to RFID solutions.

As a result, the ASPIRE middleware incorporates reader and tag virtualization capabilities, which do not rule out any reader or tag from being used in conjunction with the ASPIRE middleware. Moreover, the diversity of deployment environment dictates the high efficiency of the various components in the same fashion that the end user applications require clean and intelligently processed information. Therefore, clean and intuitive interfaces need to be throughout the whole information flow. This is achieved primarily by carefully implementing widely adopted standards (by EPCglobal) that essentially regulate the RFID domain.

The following is a list of the core ASPIRE components that comprise the middleware. These developed components are publicly available through the project’s community forge at OW2 (http://forge.ow2.org/projects/aspire/). The modular nature of the middleware architecture enabled in most cases the independent development of these components and their integration into the middleware after they reach a certain maturity level.

- Tag Data Translation (TDT)
- Reader virtualization
- Filtering and Collection (F&C)
- Business Events Generator (BEG)
- EPC Information System (EPCIS)
- Object Naming Service (ONS)
- Business Intelligence
- Connectors

These components carry out the task of transforming RFID reads into information suitable for consumption by human and/or third party software. Starting from how identification is stored on the tags (TDT), we describe how the various readers that can comprise a large-scale RFID installation can be controlled (reader virtualization). We describe the layer responsible (F&C) for performing first-level processing in the data captured by the readers, its detailed technical characteristics concerning and its interface towards the next level, the BEG component that encloses the business logic of an application. Next, the EPCIS repository is

explained, as well as the naming services it offers (ONS), the business intelligence component and the connectors that operate conceptually on top of the EPCIS.

We have to underline that in parallel, the ASPIRE Programmable Engine (APE) was developed that, combined with the ASPIRE Process Description Language (APDL) can configure the middleware in order to provide fully functional RFID solutions. These two components are described analytically in the ASPIRE Public Deliverable D4.2. Also, among the main contributions of the middleware is the creation of the BEG component, able to extract business intelligence from RFID events.

Moreover, numerous modifications and bug-fixes have been committed back to licensed open source projects that were used. For instance, the interaction between ASPIRE and Fosstrak led to benefits for both parties creating in the same time added value for the RFID community. This way ASPIRE has become part of the RFID ecosystem. In addition to the open nature of the components, the distribution network developed in parallel enables the creation of an open community which has leads to a wider adoption by users and developers.

Compared to the interim version of this document, more complete details for each component are provided since the respective development efforts led to more mature components with relatively stable technical characteristics and behavior. We focus on the respective functional, implementation and configuration details and the interfaces offered to the other components or to third party software.

Section 1 Introduction

The use of RFID is gaining an increasing popularity during the latest years, in a wide range of fields including for instance asset tracking, person identification, retail stores. The applications that support the use of RFID are the first step towards the realization of long awaited visions such as the Internet of Things. Fortunately, RFID technology is based on simple operational principles. However, the complete design, implementation and deployment of any non-trivial RFID solution requires a great effort in orchestrating the various components that comprise the architectural approach. Indeed, non-trivial RFID applications comprise typically multiple readers and tags, as well as multiple consuming applications in typically heterogeneous environments.

At a high level of abstraction, an RFID solution has to route different tag information streams across different business applications, according to sets of complex business rules. Given the complexity associated with such a task, the development of RFID solutions is nowadays facilitated by middleware infrastructures which, in short, undertake to interface to heterogeneous readers, filter the tag streams, generate application specific events, and eventually route these events to the appropriate business applications.

Placed in this context, the main goal of ASPIRE is to develop a royalty free, open source, programmable middleware platform for building RFID solutions. This platform is expected to facilitate European companies in general and SMEs in particular to develop, deploy and improve RFID solutions. In-line with its open-source nature this platform aims at offering immense flexibility and maximum freedom to potential developers and users of RFID solutions. This versatility includes the freedom of choice associated with the RFID hardware (notably tags and interrogators), which will support the solution and the software that will consume the information generated by the ASPIRE middleware.

In this deliverable we will provide information regarding the architecture of the ASPIRE middleware and the development of the components it comprises. The structure of this document is as follows:

- Section 2 provides a detailed description of the components that comprise the ASPIRE middleware architecture
- Section 3 provides details on the application servers that host these components
- Section 4 provides details on the tag data translation component
- Section 5 provides implementation details on the components that will handle the readers and sensors
- Section 6 provides development details on the filtering and collection component, which is responsible for the filtering of the raw data
- Section 7 analyzes the details of the business event generator component that handles higher level data reports and add business logic
- Section 8 provides details on the EPCIS components which are responsible for the storage and provision of the processed information
- Section 9 gives details on the ONS components which is acts as a name service in the network of things that the middleware establishes
- Section 10 provides information on the business intelligence component which is responsible for delivering reports over an RFID network

- Section 11 provides information regarding the connector component which is responsible for connecting the middleware with the actual consumers of the information; the legacy IT systems.
- Section 12 sums up our main observations and conclusions
- Section 13 presents the main issues that remain yet to be investigated in future steps

Finally, in the appendices, more technical details and source code snippets are provided in order to provide a clearer view on the implementation characteristics.

Section 2 Overview of the core ASPIRE middleware infrastructure

As stated in [16], the ASPIRE middleware platform aims at providing an effective method for SMEs to deploy RFID with a significantly lower entry cost and without the need to engage extensively with low-level middleware. In order for the middleware to accomplish this target, it should be designed and built in a transparent way for the end-users. This transparency will enable end-users and legacy enterprise systems to exploit the services of the RFID sensor system in a non-obtrusive manner. The miscellaneous components of this black box should be as much aligned as possible to the standards so that this middleware will not end up as another proprietary solution.

The core ASPIRE middleware infrastructure provides the entire functionality upon which the ASPIRE IDE and tools are designed and intended to work on. It is responsible for handling the following:

- Low level RFID and sensor readings
- Filtering and collection of the readings based on programmable filters
- Translation of the filtered data to meaningful business events
- Storing the business events
- Making available the Business events to ASPIRE or 3rd party applications
- Providing end-to-end management for the entire core ASPIRE middleware infrastructure

The high level middleware architecture is depicted in Figure 1 below.

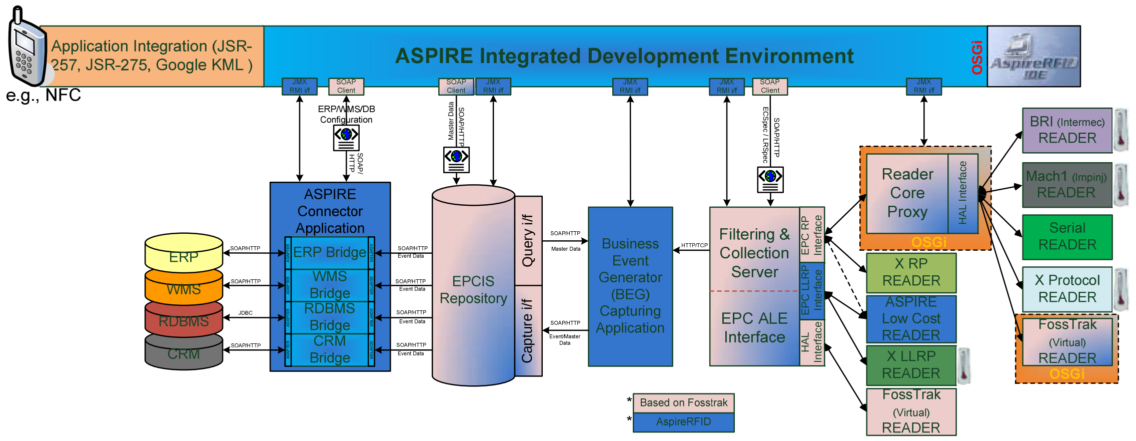


Figure 1 ASPIRE Middleware Architecture

Each node in the architecture represents a hierarchical level of functionality, starting from the hardware level and provides a functional abstraction to a conceptually lower hierarchical level.

The conceptual hierarchy that is imposed in the middleware architecture starts from the hardware level, which contains all the required hardware with its proprietary APIs. At a higher level the Hardware Abstraction Layer (HAL) is introduced, which hides the proprietary communication aspects of the hardware from the higher levels. The event level utilizes the abstraction provided by the HAL and processes the streams of data from the hardware level [1]. The outcome of this process is information about low level events.

Although low level events are fewer than the raw RFID reads, they still are significant in amount and do not provide business level information. The role of leveraging their content into the business event layer is handled by two components: first the Filtering and Collection (F&C) component and next the Business Events Generator (BEG) component. These two components act in a complementary manner transforming the lower level events into business events. This transformation is only possible with the provision of additional metadata, which are appropriately handled by the BEG component.

This information (i.e. business events) is then forwarded to a higher hierarchical level, where it is consumed by the Information System (IS) component. The IS component comprises a repository (i.e. a database) which aggregates events received by the lower levels, applies additional business logic and stores business information, which could then be conveyed to the company's enterprise IT systems (e.g., a Warehouse Management System (WMS), an Enterprise Resource Planning (ERP) system or corporate database repositories). Hence, well-defined connectors should drive the integration of information sharing repositories with enterprise business systems. The connectors are also in charge of exchanging information between the IS repositories and the business systems using data-centric (e.g., direct data access) and/or messaging mechanisms (e.g. EDI, XML messaging, Web Services) mechanisms. Moreover, in the scope of open loop systems this information can be provided to other business partners, through either the enterprise systems or the information sharing repositories.

Apart from the architecture functional plan, a management plan whose task is to manage and orchestrate the subsequent components presents in the overall middleware infrastructure. The management plan ensures that the middleware components comprising an ASPIRE system operate appropriately, while at the same time provide a functionality for runtime module management (e.g., starting, stopping, deploying and (re)configuring components).

Under this prism, the following Sections drill down to the various architectural nodes and analyse their functionalities and interfaces.

Section 3 Infrastructure components

3.1 Application server

The components of the ASPIRE infrastructure are hosted on two types of servers.

First, the Apache Tomcat Web Server and servlet container is responsible for hosting key components in the ASPIRE Architecture: The Application Level Events (ALE) server and the EPCIS clients implementing Capture and Query interfaces. Apache Tomcat is an open-source web server implementation in Java, freely available under the Apache License (version 2). The software has been tested and verified that it runs well under Tomcat versions 6.x.

Secondly, the OSGi server Apache Felix is used in order to host the Reader Core Proxy. Apache Felix is an open-source implementation of the OSGi R4 Service Platform and other OSGi-related technologies, available under the Apache license. The OSGi Services platform [14] is a framework for the development of modularized Java applications using a Service Oriented Architecture approach inside the same Java Virtual Machine. The OSGi specifications originally targeted embedded devices and home services gateways, but they are ideally suited for any project interested in the principles of modularity, component-orientation, and/or service-orientation. Although the effort has been started in 1999, the software industries (e.g. Eclipse, Jonas, Glassfish) have just recently shown a trend to adopt it as a component platform. Among the goals of OSGi are provisioning type isolation between components, hot deployment of components (i.e. installation, updates without needing to shut down or reset the application), and a loosely coupled service based interaction between these components. By separating a system in different and decoupled components, there is a major advantage for system maintenance and evolution. In addition, this infrastructure allows increasing the uptime of applications since it is possible to replace components on the fly.

We need to note that the implementation efforts by UJF (RFID Suite) and AspireRfid are now merged in the OW2 subversion repository. The current implementations are no longer being maintained separately but they are all part of the AspireRfid middleware project. However, as the edge and premise servers of the RFID Suite are implemented using OSGi technology, it was necessary to evaluate which concepts could be brought to the main AspireRfid branch. Section 6.5 provides more details on the concepts that have been implemented as OSGi applications.

A plugin was developed for the ASPIRE IDE, acting as a high level management console that provides a Graphical User Interface that can control the Reader core proxy, allowing operations like starting, stopping and resetting the reader core proxy, changing configuration parameters and so on.

3.2 Interfaces to other components

In the OSGi service registry, interfaces to other components are exposed as registered services. Service dependency management is done via the Apache Felix iPOJO [15] component model. Communication between different OSGi applications (edge and premise) has been done using a message based middleware (Java Message Service) as well as Web Services and SMTP.

We need to specially reference Java Management eXtensions (JMX), a standard Java API for the management and monitoring of Java applications. Its functionality resembles the SNMP (Simple Network Management Protocols) used for the management and monitoring of network equipment. By developing our JMX probes (MBeans) and installing it in the running Java application, we can expose management interfaces to the AspireRfid middleware for accessing or changing application configuration at runtime in a distant management fashion. Several AspireRfid components expose JMX interfaces in the Reader Core proxy, which could be accessed by other applications. New readers may also expose JMX interfaces so they can be configured remotely.

The EPCIS has functionality exposed as HTTP Web Services which allow it to be interrogated about tag history and tag information. This interface enables the data to be accessed by the ONS during the interrogation process as well enabling other applications (e.g. graphical interfaces for data monitoring) not necessarily developed in Java to access that information in a interoperable manner.

The ONS component, as well as the EPCIS, has some Discovery Services functionality embedded. ONS was developed using a Web Services approach (see Section 9). The goal of using an application server for deploying and hosting such applications was to take advantage of the scalability and robustness provided by Java Enterprise technology.

Section 4 Tag Data Translation

4.1 Overview and purpose

The EPC Network defines standards going from tag data to Application Level Events (ALE) in its architecture framework. These events are used by clients to obtain EPC data from sources.

The Tag Data Standard (TDS) defined by EPCglobal lies at the lowest level. Tag data is retrieved according to the Low-Level Reader Protocol (LLRP) which specifies how readers can communicate with tags using the Tag Protocol. A standard for reader management is also defined. All these standards aim to unify the way of identifying uniquely items and manage compatible readers. EPCglobal defines an interface for EPC Information Service (EPCIS) in order to share EPC-related data in and between enterprises. This standard includes EPCIS Data Specification providing definitions for all types of EPCIS data, and EPCIS Query Interfaces defining the way for querying and delivering data from EPCIS. Another standard is used for retrieving EPC-related data. The Object Name Service (ONS) provides a way of finding an EPCIS which contains the needed data.

Each of these standards uses its own format. Indeed, the data reads from tags can be used by ALE, EPCIS, legacy applications, ONS or for writing information in tags. In any case, the product code must be translated into different formats to be used in each of these scenarios, and this task is performed by the Tag Data Translation (TDT) component.

The translation of different tag data formats can be performed at any level of the RFID middleware architecture, thus the TDT is a very important tool in the EPCglobal Network. The tag-encoding URI representation of an EPC is used by the ALE. The pure-identity URI is defined for EPCIS. The EPC must be in its ONS hostname representation to perform an ONS query. The legacy application prefers to use the legacy representation of an EPC. At last, the binary format can be used for communicating with the reader (for the writing action).

The ASPIRE Tag Data Translation (TDT) [1] provides a way to convert not only EPC tags, but also other identification standards such as GS1 bar codes, ISO smart cards and tags (ISO 14443 and 15693), etc.

Representation	Value
TAG-ENCODING URI	urn:iso:tag:15693-64:98.104197
PURE-IDENTITY URI	urn:iso:id:15693:98.104197
ONS HOSTNAME	104197.98.15693.onsiso.com
LEGACY	iso15693;mfgcode=98;serial-104197
BINARY	11110000001100010000000000000000 00000000000000011001011100000101

Table 1 ISO 15693 Tag Data representation

4.2 Interfaces to other components

The ASPIRE TDT is a standalone tool that can be used at any level of the ASPIRE architecture. It is currently used at the Reader Core Proxy (translation from BINARY to TAG-ENCODING URI and vice versa) and the F&C server so as we are able to use the EPC architecture and specifications up to the EPCIS and ONS layer (with translation into PURE-ENCODING URI and ONS HOSTNAME). The TDT will be used at the Connector layer to decode the captured events from the EPCIS repository and translate EPC into the LEGACY representation that ERPs and WMSs "understand".

4.3 Current implementation status

The ASPIRE TDT is available in its 0.4 version. In this version, the TDT supports the translation into various formats of EPC tags, ISO15693, ISO14443, NFC, MAC address, phone numbers (with company prefixes) and various GS1 formats (EAN/UPC, GS1 DataBar, GS1-128, ITF-14, GS1 DataMatrix, and Composite Component).

The ASPIRE TDT takes the ID as an input, encoded in any format, and a variable number of parameters depending on each other:

(i) desired output format
(ii) input data type (GS1, ISO or PHONE)
(iii) GS1 SI and code length (for output format GS1_AI_ENCODING)
(iv) tag length, company prefix length, filter and country prefix length (for some input format)

Figure 2 shows the TDT behavior. We use the Fosstrak implementation of the EPC TDT as core of EPC translation.

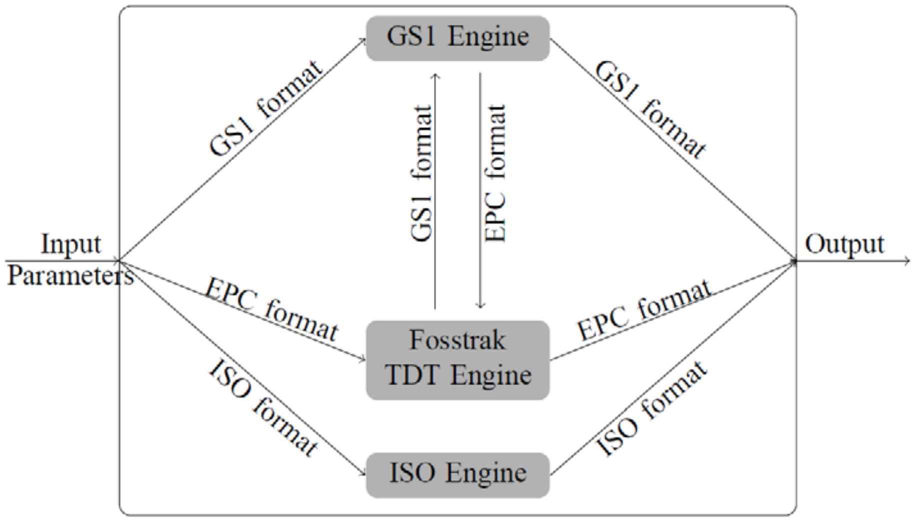


Figure 2 ASPIRE TDT Engine

Technical Requirements:

- JDK: 1.6.0_05 or higher
- Memory: No minimum requirements.
- Operating system: No minimum requirements. The component is also tested on Linux

4.4 Summary

The tag data translation engine roadmap is summarized in the table below:

Tag Data Standars	Supported
Bar Code Tags (GS1 System)	Yes
EAN/UPC	Yes
ITF-14	Yes
GS1 DataMatrix	Yes
GS1 DAtaBar	Yes
GS1-128	Yes
EPC Global Tags	Yes
SSCC	Yes
GTIN	Yes
GLN	Yes
GRAI	Yes
GIAI	Yes
GLN	Yes
ISO Tags	Partially supported
14443	Yes
15693	Yes
15962	Yes
Others IDs	Partially supported
NFC	Yes
Mac address for bluetooth and zigbee sensors	Yes
Phone number (with country prefix) for cell phones	Yes
Device id for OneWire devices (iButton)	Yes

Table 2 Summary of the identifier types supported by the TDT

Additionally, several changes and bug fixes were implemented and contributed back to the Fosstak TDT library. Table 3 below summarizes these changes and bug fixes.

S/N	Description/Modules and Rationale
	Enhancements:
1	Made implementation also OSGI compliant and deployable
2	Provide TDT engine as OSGI service
3	Expanded the Tag types support from Current EPC Global Tags to support also: • Bar Code Tags (GS1 System) • EAN/UPC • ITF-14 • GS1 DataMatrix • GS1 DataBar • GS1-128 • ISO norms (15961, 15962, 15963, 15693, 14443, 18092) • MAC addresses for bluetooth and zigbee sensors • Phone numbers

	With the Extended version of TDT the F&C module will be able to support more devices (e.g. barcode readers …). The information will be able to be filtered from the Fosstrak F&C server, generate the required events from the AspireRFID BEG, store them to the Fosstrak EPCIS repository and finally translate them again back to their original state at the AspireRFID Connector level to deliver them to an ERP or WMS.

Table 3 Changes and bugfixes committed back to the Fosstrak TDT library

Section 5 Reader interfaces

5.1 Low Level Reader Protocol Interface

5.1.1 Overview and purpose

The LLRP protocol enables the ASPIRE middleware to communicate with LLRP protocol compliant readers. The LLRP [6] is a standard specification defined by EPCglobal to allow standardized communication with RFID readers. This communication involves interacting with RFID tags and low level configuration of the reading devices. The main difference between LLRP and the simple Reader Protocol (RP) is that the former is fully aware of the RFID air while the latter was designed to provide a higher level abstraction that hides many RFID details.

This interface has been fully integrated to the ASPIRE middleware through the use of the open libraries provided by the open source project LLRP-Toolkit. The LLRP Toolkit houses the development of open source libraries in various languages to help reader and software vendors build and parse LLRP messages [7].

5.1.2 Interfaces to other components

The LLRP interface is part of the Application Level Events (ALE) server. It enables this component to interact with LLRP compliant readers using standard LLRP messages.

5.2 Supported Readers

5.2.1 Overview and purpose

UJF has already develop drivers for several RFID readers such as TagSys Medio L100, TI Tiris 6350, ACR 122 (i.e. Touchatag), Violet Mirror in the OSGi branch. Moreover, UJF has already developed reader drivers for other identification technologies such as DS iButton. UJF has already developed a bridge between HTTP and Bluetooth to collect tag events from HTTP client (i.e. legacy software in Java or other languages such as C# or C) and Bluetooth clients (i.e. NFC phones).

5.2.2 Interfaces to other components

Reader drivers are packaged as OSGi bundles manageable with JMX MBean interface. The bundle containing the driver uses the Event Admin service to publish tag events to the F&C bundle or others (an Event-Condition-Action (ECA) rule engine for instance). The current implementations rely on the Event Admin service.

5.2.3 Rifidi Interface

The middleware was enhanced with integration capabilities with the widely known Rifidi Designer application (available online at www.rifidi.org).

Rifidi designer is an open-source application that allows the modeling and simulation of RFID processes. Through its GUI, the user can setup a virtual environment, define readers, interrogation zones and RFID tags and then configure their respective behaviors. Using the application, the user can assign specific roles and paths that the constituent components of a scenario can follow. In addition, there is a possibility to model the area where the system will be deployed (simulated). The authors claim that the physical aspects of an RFID deployment have been taken into account to a certain extent. It is also possible to automate simulations of various scenarios. Therefore, the Rifidi designer is a powerful tool in the hands of a system designer/developer or researcher on the domain.

For these reasons, and in order to allow AspireRfid tests to be conducted in a more automated way, integration efforts led to the resulting implementation of an LLRP interface through which the AspireRfid middleware can communicate with a Rifidi Designer modeled scenario. In particular, the LLRP interface was developed as an enhancement at the F&C layer.

Figure 4 below gives an overview over the logical architecture followed by the integration. As it can be observed, in the use case depicted, 3 LLRP Virtual readers were created in the environment of the Rifidi designer. All three virtual readers produce readings according to the modeled scenario. These readings can be then further used by ASPIRE in order for instance to conduct theoretical measurements or to showcase and study a system's behavior without the necessary presence of any hardware infrastructure.

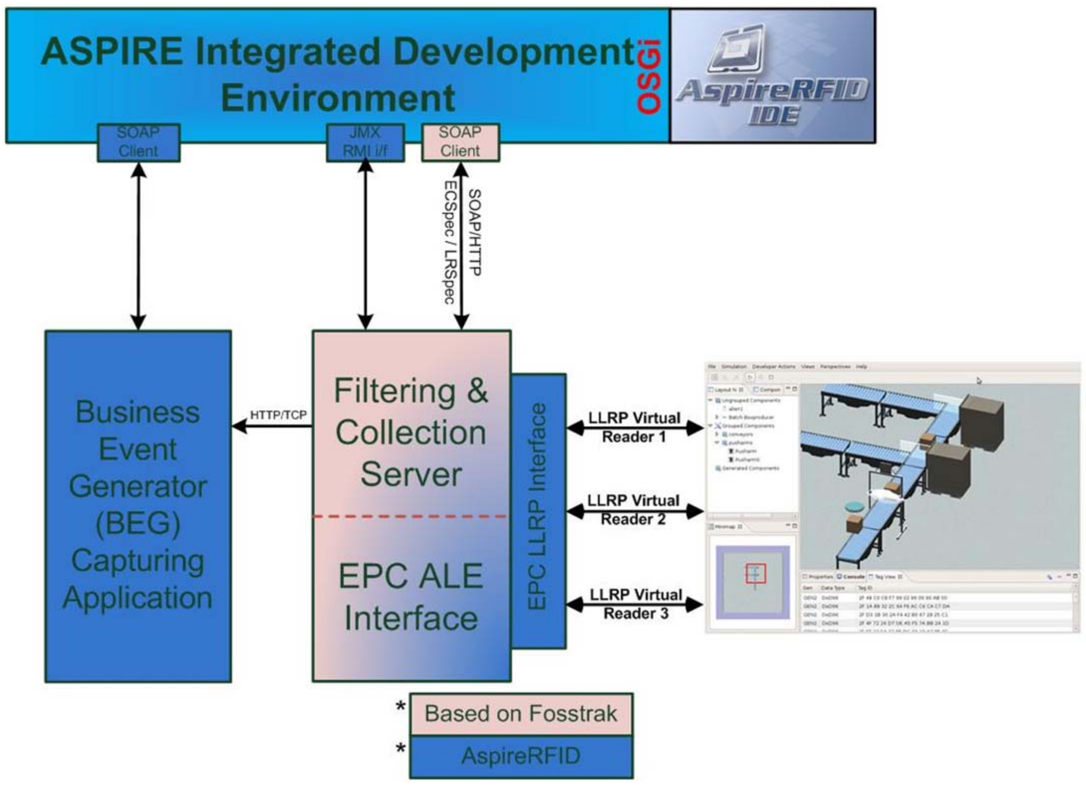


Figure 3 Integration with Rifidi Designer

5.2.4 Fosstrak Capture Interface (AIT)

Also, the following have been contributed back to Fosstrak, regarding the Reader Core module and the HAL, as analyzed in Table 4.

S/N	Description/Modules and Rationale
	Reader Core Module:
	Enhancements:
1	Made implementation OSGI compliant and deployable
2	Made configuration dynamic through JMX
3	Enabled management through JMX (start/stop/restart/reset modules)
	BugFixes:
4	Add tags to the Tag To Report Group list as "Pure URI"
5	Use of 96 bit tags from 64 supported in the code
	Reader HAL:
	Enhancements:
6	Development of Hardware Abstraction Layer for INTERMEC readers (at the reader core layer)
7	Made implementation OSGI compliant and deployable
8	Made configuration dynamic through JMX
	BugFixes:
9	Fixing a bug in the Hardware Abstraction Layer for Impinj readers (bug at using more than one antenna)

Table 4 Reader Core and HAL enhancements and bug fixes committed back to Fosstrak

Note that due to the fact that mainly the execution environment has been changed at the Reader Core the two "flavors" of the Reader Core module could co-exist. The one could be a Fat Client (which already exists) and the other could be a version which runs within an OSGi container.

5.2.5 Portable reader interface

Communication with a portable RFID reader (Scemtec) is implemented through LLRP messaging. Using this HAL, a portable RFID server can be used with the middleware. As illustrated in Figure 4 below, Scemtec commands/responses can be "translated" to LLRP commands/responses and vice-versa.

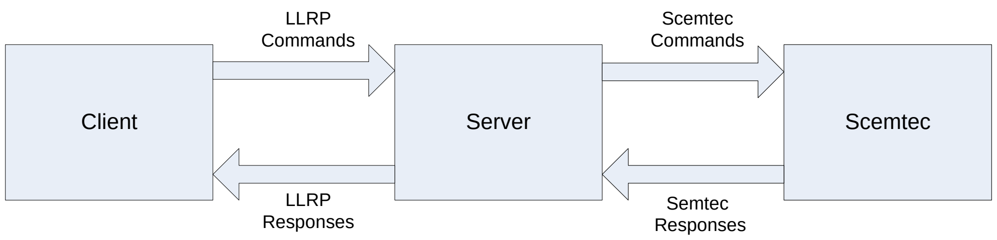


Figure 4 Communication with a portable RFID server through LLRP-Lite

The thin LLRP Server / Proxy (LLRP Lite implementation) is suitable for resource constrained environments.

5.2.6 Active Tags (AAU)

Active RFID has some advantages compared to passive RFID, such as long reading range (30-100+ m). Moreover, the active tag integration with sensor is possible and widely available in the market. Hence enables its usage for some particular applications that need tracking of sensing information or real time tracking, and operate in large area with severe environment (with respect to the wireless channel). According to this fact, the support to the active RFID in middleware is one of the important issue. The integration of active RFID into AspireRFID Middleware consists of two main tasks in the development point of view, i.e. HAL implementation of active RFID reader and active tag data support in the middleware, including the sensing data.

5.2.6.1 Active Reader HAL implementation

The HAL implementation is based on active reader from GaoRFID manufacturer [17]. The reader operates in 2.45 GHz frequency band. Its antenna gain can be adjusted with 32 gain level where the maximum reading range of the highest gain is claimed to be up to 100 m. Moreover, currently it has three types of tags, i.e. tag with temperature sensor, tag with vibration sensor, and tag without sensor. The HAL is implemented such that it is able to configure the antenna gain, reader reading mode, and carry the sensing data as well. This HAL is part of ALE server and is connected with the F&C layer. It enables this component to interact with active reader.

The configuration of antenna gain and reader reading mode as mentioned earlier is done in the LRSpec. The reader reading mode here is defined as a way the reader reads the data from the tags and forwards the data to the client application or middleware. The reader can work in two following reading modes:

- Direct mode: the data is forwarded directly upon receiving RFID signals from the tag,
- Buffer mode: it means that the last 100 bytes data from tags is saved in buffer and will only be sent upon receiving read buffer command from the host, i.e. middleware.

In the LRSpec, antenna gain is set to be an integer value ranging from 0 to 31, while the reader reading mode is set to be "FF" for direct mode and "0" for buffer mode. Figure 5 depicts the LRSpec of the integrated active RFID reader.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns3:LRSpec xmlns:ns2="urn:epcglobal:ale:wsdl:1"
xmlns:ns3="urn:epcglobal:ale:xsd:1">
    <isComposite>false</isComposite>
    <readers/>
    <properties>
        <property>
            <name>Description</name>
            <value>This Logical Reader is serving the active Tag in
2.4GHz band</value>
        </property>
        <property>
            <name>ConnectionPointAddress</name>
            <value>10.110.11.107</value>
        </property>
        <property>
            <name>ConnectionPointPort</name>
            <value>10001</value>
        </property>
        <property>
            <name>ReadTimeInterval</name>
            <value>1000</value>
        </property>
        <property>
            <name>PhysicalReaderSource</name>
            <value>1</value>
        </property>
        <property>
           <name>ReaderGain</name>
            <value>0</value>
        </property>
        <property>
           <name>ReaderGainMode</name>
            <value>171</value>
        </property>
        <property>
           <name>ReaderMode</name>
            <value>FF</value>
        </property>
        <property>
            <name>ReaderType</name>
            <value>
org.ow2.aspirerfid.ale.server.readers.gaorfid.GaorfidAdaptor</value>
        </property>
    </properties>
</ns3:LRSpec>
```

Concerning the sensing data, in this case it is included in the RFID packet data from the tag. Therefore the HAL is designed such that it is able to extract its value with the help of new data fields; afterwards the sensing data is included into the tag object that is being notified to the Event Cycle.

5.2.6.2 *Integration of active tag data and sensing data*

Currently the integration of active tag data and sensing data has been implemented in the F&C layer, thus it enables the middleware to filter particular tag id and collect the sensing data at the same time. The integration of sensing data involves the modification of ECSpec and ECReport by adding the “extension” field, as well as the behavior of ALE server upon generating the report. The implementation is done based on Foostrak library.

The ECSpec is modified by adding an extension field in the reportSpec field. The filterSpec is also defined inside the reportSpec because the sensing data comes together with the tag ID. Furthermore, it will make sure that the received sensing data is related with a certain tag ID when generating the ECReport. The code below depicts part of ECSpec where extension of sensing data is inserted.

```
<reportSpec reportOnlyOnChange="false" reportName="temperatureTest"
reportIfEmpty="true">
            <reportSet set="CURRENT"/>
            <filterSpec>
                <includePatterns>
                    <includePattern>urn:epc:pat:gid-
96:0.3003.*</includePattern><!-- temperature tag -->
                </includePatterns>
                <excludePatterns/>
            </filterSpec>
            <groupSpec/>
            <output includeTag="true" includeRawHex="true"
includeRawDecimal="true" includeEPC="true" includeCount="true"/>
         <extension>
                <extension application="temperature" sensorID=
"urn:epc:pat:gid-96:0.3003.*" senseValue="0" error="0"
unit="celcius"/>
         </extension>
</reportSpec>
```

The code below depicts an example of ECReport which present the sensing data together with tag ID.

```
<report reportName="temperatureTest">
      <group>
      <groupList>
           <member>
                <epc>urn:epc:id:gid:0.3003.47244640358</epc>
                <tag>urn:epc:tag:gid-96:0.3003.47244640358</tag>

<rawHex>urn:epc:raw:96.x350000000000BBBB00000066</rawHex>
                <rawDecimal>
urn:epc:raw:96.16402705520531495054246674534</rawDecimal>
                <extension unit="celcius" application="temperature"
error="0.0" senseValue="22.0"
sensorID="urn:epc:id:gid:0.3003.47244640358"/>
                </member>
            <member>
                <epc>urn:epc:id:gid:0.3003.47244640360</epc>
                <tag>urn:epc:tag:gid-96:0.3003.47244640360</tag>

<rawHex>urn:epc:raw:96.x350000000000BBBB00000068</rawHex>
                <rawDecimal>
urn:epc:raw:96.16402705520531495054246674536</rawDecimal>
                <extension unit="celcius" application="temperature"
error="0.0" senseValue="22.5"
sensorID="urn:epc:id:gid:0.3003.47244640360"/>
```

```
                </member>
            </groupList>
        <groupCount>
            <count>2</count>
            </groupCount>
    </group>
</report>
```

5.3 NFC

5.3.1 *Overview and purpose*

Near-Field Communications (NFC) covers short distance communications between RFID tags (ISO 14443A/B, Felica, MiFare) and fixed or mobile readers. Since the NFC had already met the mass market in Japan (50 million NFC-enabled phones and 7 million NFC-enabled PCs mid 2009) it will probably meet it in developed countries in the coming years.

NFC Forum defines specifications [12] for Near-Field Communications (NFC) applications such as product information, smart posters, discount vouchers, ticketing and payment. The specifications are limited to the list of supported RFID tags standards and products and to the information stored in the tags. The forum has not yet defined a global architecture similar to the EPCglobal one in order to integrate NFC information in companies' information systems. The JCP (Java Community Process) has specified an API for contactless communications (JSR 257) [13]. This API is mainly led by Nokia and enables to develop J2ME applications using NFC tags.

5.3.2 *Interfaces to other components*

The OSGi branch provides already a library of SW components to ease the building of NFC MIDLets (ie J2ME applications using the JSR 257). The NFC MIDLet can search a ASPIRE BlueTooth Bridge (BTB), bind it and send tag events to the BTB. The BlueTooth Bridge (BTB) is currently ALE event publisher converting and sending events to the F&C layer.

Section 6 Filtering and Collection

6.1 Infrastructure

6.1.1 Overview and purpose

The filtering and collection component is a very significant component in the ASPIRE middleware architecture. It is the implementation of the ALE standard [8] which specifies an interface through which clients may interact with filtered, consolidated EPC data and related data from a variety of sources. The design of this interface recognizes that in most EPC processing systems, there is a level of processing that reduces the volume of data that comes directly from EPC data sources such as RFID readers into coarser "events" of interest to applications. It also recognizes that decoupling these applications from the physical layers of infrastructure offers cost and flexibility advantages to technology providers and end-users alike.

In the scope of large scale deployments, RFID systems generate an enormous number of object reads. Many of those reads represent non-actionable "noise." To balance the cost and performance of this with the need for clear accountability and interoperability of the various parts, the design of the ASPIRE middleware seeks to:

- Drive as much filtering and counting of reads as low in the architecture as possible.
- Minimize the amount of "business logic" embedded in the Tags.

6.1.2 Interfaces to other components

The Filtering and Collection Middleware is intended to facilitate these objectives by providing a flexible interface to a standard set of accumulation, filtering, and counting operations that produce "reports" in response to client "requests." The client will be responsible for interpreting and acting on the meaning of the report. Depending on the target deployment the client of the ALE interface may be a traditional "enterprise application," or it may be new software designed expressly to carry out an RFID-enabled business process, but which operates at a higher level than the "middleware" that implements the ALE interface.

In the scope of the ASPIRE project, the Business Event Generation (BEG) middleware would naturally, consume the results of ALE filtering. However, there might be deployment scenarios where clients will interface directly to the ALE filtered streams of RFID data.

6.1.3 Current implementation status

The ASPIRE Filtering and Collection component is a modified version of Accada RFID Middleware, which is now called Fosstrak [9], licensed under LGPL. The ASPIRE F&C consortium has worked and implemented changes and bug fixes on the original Accada middleware. These changes and bug fixes are outlined in Table 5 below.

S/N	Description/Modules and Rationale	
		Enhancements:
1	ASPIRE has created two EPC LLRP Interfaces (to support LLRP compliant RFID readers) with the use of LLRP toolkit V0.1.1. The configuration of these interfaces is done directly from the LRSpec definition. The one interface programs the reader to produce reports at a stable time interval (defined by the user) (supports Rifidi	

	Designer). The other interface's timings are controlled by the F&C module and can be modified on the fly by triggering (supports Rifidi Designer and to be used for multi-vendor LLRP anti-collision protocol). • The use of such interfaces, whereas they do not give as much control as the Fosstrak's LLRP Commander, they are faster to be configured because they don't require extra tooling and extra configuration steps. Our proposal is, for the Filtering & Collection module to support many LLRP HAL's so as the user/developer to be able to choose the one that is more suitable for his application and needs.
2	ASPIRE has created a HAL interface at the F&C layer for the TagSys Medio L100 L200 + HF RFID reader.
3	The EPC ALE specification requires for the Filtering and Collection module to be able to include at its reports the following Tag format types: Tag, RawHex, RawDecimal and EPC. So except from EPC format that was supported, the following changes were made to support the rest of them. • Added functions for converting byte[] Tag IDs to raw Decimal, raw HEX and tagURI formats so as the produced ECReport to be able to include these Tag format types. • Add check at the Pattern Class method isMember(String tagPureURI, String tagURI) to check whether the requested pattern from the ECSpec is pure id pattern or not so as to use the right Tag format (Pure URI or Tag URI). • Add captured tags to the Tag Report Group list as: o Raw Decimal o Tag URI o And Raw Hex
4	Added the ability to clear the ADDITONS and DELETIONS history at defined amount of event cycles as specified at the EPC ALE 1.1 specification.
5	For now the Filtering and collection module supports only GID-96, SGTIN-64 and SSCC-64 Tag type. For extending the Filtering and Collection server ASPIRE have added SGTIN-96, SGTIN-198, SSCC-96, GSGLN-96, GSGLN-195, GRAI-96, GRAI-170, GIAI-96, GIAI-202, USDOD-96, GID, SGTIN, SSCC, GSGLN, GRAI, GIAI, USDOD to the "PatternType" Object for future support of the rest EPC Tag types.
6	ASPIRE have updated Apache CXF library for the web service implementation from 2.0.4-incubator to 2.2.4
7	The "Universal" common library is used instead of fc-commons
	BugFixes:
8	The attributes "value", "low" and "high" at the PatternDataField.java was changed to long from int because e.g. the gid-96 serial number can get up to 11 digits and int supports up to 10 and that was causing an exception to be thrown and the filtering procedure to stop when a tag with 11 digit serial number was received.
9	Add third field check to the URI pattern (Range is not allowed in pure id patterns as required from the ALE Specifications).
10	Clear the faulty read tags (e.g. Invalid URI pattern) in the ECSpecValidationExceptionResponse and the ImplementationExceptionResponse so as Filtering and Collection server implementation wont "stuck" if an exception occur. All previous tags will be deleted so ADDITIONS and DELETIONS will be reset.
11	At EventCycle.run() corrected the faulty use of "lastEventCycleTags=tags" which was replacing the hole list of the Tags read with the new list of Tags read and

	changed into "lastEventCycleTags.addAll(tags)" so as not to lose the previous collected tags before the event cycle finish and the Tags are reported.
12	Fixed faulty behavior when using ADDITONS and DELETIONS in the same ECSpec (previously if the same tag group was used in two Report Specs, one for ADDITONS one for DELETIONS, it was adding and deleting tag ids at the ECReports arbitrarily).

Table 5 Changes and bug fixes in the Fosstrak ALE module

6.2 Anti-collision protocol

6.2.1 Overview and purpose

In a RFID network while several readers are placed close together to improve coverage and consequently read rate, reader-reader collision problems happen frequently and inevitably. High probability of collision not only impairs the benefit of multi-reader deployment, but also results in misreading in moving RFID tags. Indeed, when a tag enters an area covered by more than one reader, it will be detected by none reader because of reader-reader collision. In order to eliminate or reduce reader collision, we propose an Adaptive Color based Reader Anti-collision Scheduling algorithm (*ACoRAS*) where every reader is assigned a color that allows it to read moving tags during a specific time slot within a time frame. Only the reader holding a color (token) can read at a time. The algorithm guarantees that any moving tag that spends a minimum T_{min} in a reader coverage area is read. The algorithm ensures that neighboring readers are assigned different colors to eliminate collisions. We generalize the algorithm and allow readers to be assigned more than one color to improve performance by reducing reader idle time thus increasing throughput.

To illustrate a reader collision, let's have a look at Figure 5. The tag enters the grey area which is the range of reading of readers 1, 4 and 5. The idea here is to schedule these readers so that all these readers are not active at the same time. In addition, note that the area covered by the field of Reader 5 is covered by the sum of the fields of readers 1 to 4. In order to save energy, Reader 5 can be idle. This first step to allow Reader 5 to sleep will limit collisions that spare energy and data processing since requiring some tag filtering.

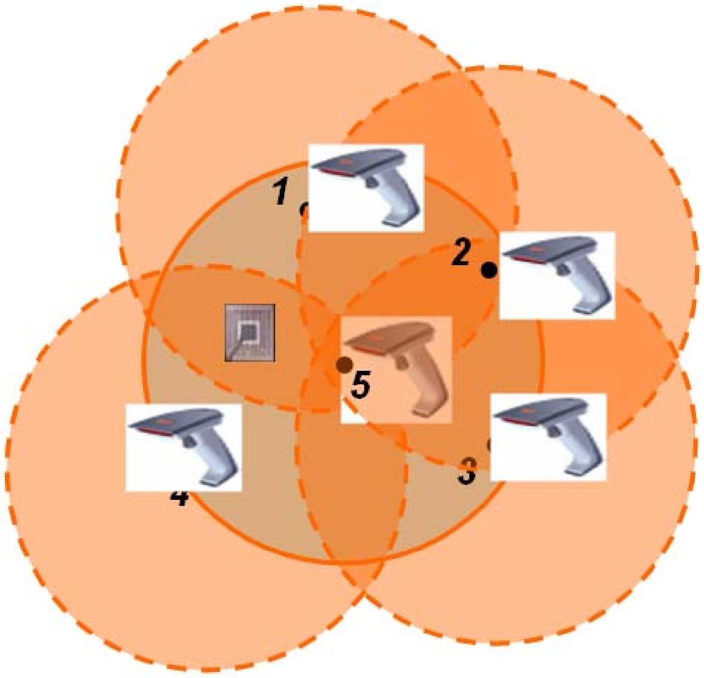


Figure 5 Example of reader collision

An alternate solution would be to schedule the readers to avoid them to work all at the same time and spend energy uselessly, i.e. a reader anti-collision mechanism. This will allow leveraging less traffic, saving bandwidth and computing resources at the ALE level. In order to reduce the memory and processing resource requirements, the reader scheduling should be achieved in a distributed and local manner, while ensuring that at any time, the whole area is covered. For instance, on Figure 5, reader 5 may remain quiet. Its area will be collectively covered by readers 1,2,3 and 4 and the RFID tag will still be detected by readers 1 and 4.

6.2.2 ACoRAS

To simulate a typical deployment of RFID device tracking and supply chains requirement, we assume that tags are mobile but readers are static. We focus on the problem of RFID reader collision and propose a reader activity scheduling algorithm that minimizes reader collision and increases throughput while respecting hardware and applications requirements. The algorithm that we refer to as *ACoRAS* (Adaptive Color based Reader Anti-collision Scheduling algorithm), is based on assigning colors or tokens to readers.

ACoRAS ensures that every tag that spends less than T_{min} seconds in the field of a reader is detected by optimizing the number of active readers and ensuring no reader collision.

ACoRAS runs in two steps. In the first step, readers are assigned colors that allow them to communicate. The number of colors a reader can hold may be greater than one (the larger the number of colors a reader holds the more chances he is given to communicate). However, due to the number of tags to be read, the number of colors and thus the induced time slot may not handle such an assignment. Indeed, when colors are too numerous, the single time slot duration may be too short to allow reading every tag laying in reader field.

Therefore, the second step aims at reducing the number of colors a reader can hold to satisfy system requirements by minimizing collision. To do so, ACoRAS removes the colors which are the less in conflict and replaces them by a color which ensures that even in collision appears at reader 1 at time t, tags laying in the field of reader 1 will be eventually read before t+ T_{min}.

Our extensive simulations show *ACoRAS* outperforms existing methods; the proportion of unread tags is less than 0.75% for various numbers of readers (consequently proportion of interference area) and various number of tags.

6.3 Distributed Filtering and Collection server

6.3.1 Design and principles

A problem arises with the actual EPCGlobal ALE standard when too many tags are read or too many readers are connected to a single ALE. In such cases, a bottleneck appears between readers and the ALE. Indeed, in our typical architecture, if a hundred readers are connected and each reader read a thousand of tags simultaneously, the network may not be able to manage so many readers events and the ALE may not be able to process so much data. So, one ALE for managing all company warehouses readers is not scalable enough. A solution would be to duplicate ALE engines. The standard offers to business applications a way to make a stocklist of all its warehouses by sending a specification to each ALE in each warehouse. All ALE engines will then send a report and the business application can draw its inventory. A problem arises here: the transparency for applications which has to split ECSpecs and merge ECReports themselves, this is not their role. A solution should be to have a Global ALE that performs these two operations [17]. By providing this transparency, a new bottleneck appears between sub-ALEs and the Global one, as shown in Figure 6.

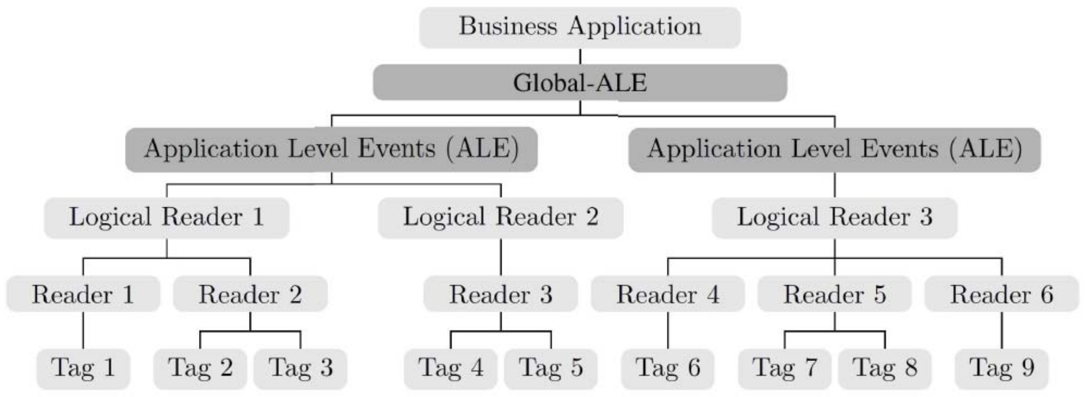


Figure 6 Global ALE Architecture

We propose a mechanism based on distributed hash tables and peer-to-peer (p2p) mechanisms that offer the same scalability to the ALE engine but in a transparent manner for business applications. An ALE, which is a node of the p2p system, receiving specification has to split and distribute it to other involved ALEs and merge all reports locally before send the final report to the business application.

Providing scalability, dynamicity, and preventing network overload, P2P systems solve some parts of our problems. The first step is to map P2P elements with RFID middleware architecture. In our solution, ALEs are nodes, and readers are objects to store and share by the system. In the ALE, readers are configured as logical reader via LRSpecs files. The configuration of logical reader is simple, a logical reader can be composite or not. As physical readers are objects to share, we use for now only non-composite logical reader.

The business application that wants to perform an inventory of two readers that not connected to the same ALE will provide an ECSpecs file with the two above-mentioned readers. The following XML file presents such an ECSpecs. First, it declares the logical readers involved in the process (lines 3-6). The boundarySpec section (lines 7-11) explains the start and stop boundary and the reportSpecs (lines 12-17) configures the ALE to report all the EPC's of tags present in the field of the reader. In other words, this specification describes an inventory operation with two readers involved, and repeats every 10 seconds for 9.5 seconds of duration.

```
1   <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2   <ns2:ECSpec xmlns:ns2="urn:epcglobal:ale:xsd:1">
3         <logicalReaders>
4               <logicalReader>Reader1</logicalReader>
5               <logicalReader>Reader2</logicalReader>
6         </logicalReaders>
7         <boundarySpec>
8               <repeatPeriod unit="MS">10000</repeatPeriod>
9               <duration unit="MS">9500</duration>
10              <stableSetInterval unit="MS">0</stableSetInterval>
11        </boundarySpec>
12        <reportSpecs>
13              <reportSpec>
14                    <reportSet set="CURRENT"/>
15                    <output includeTag="true"/>
16              </reportSpec>
17        </reportSpecs>
18  </ns2:ECSpec>
```

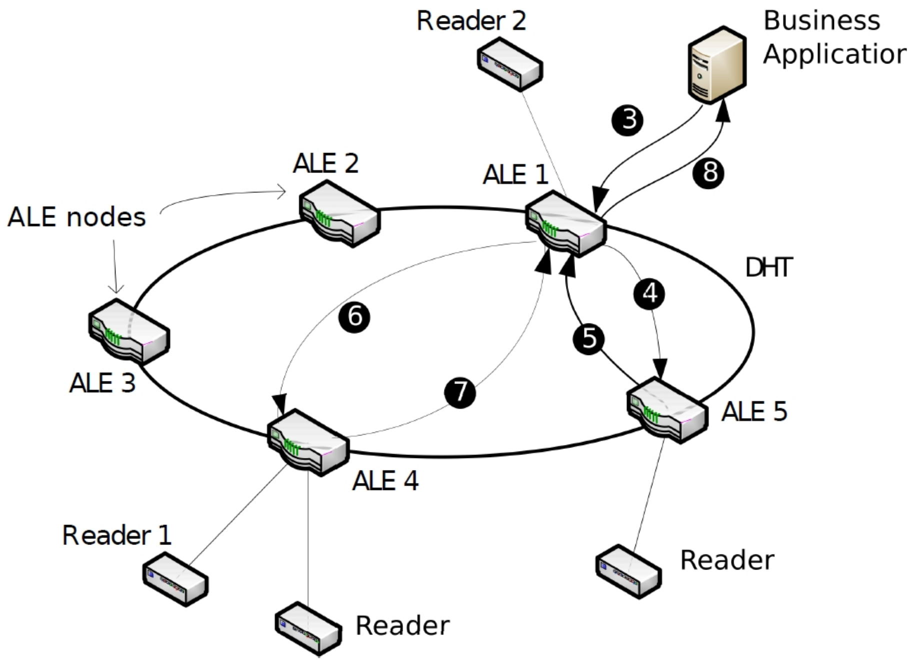


Figure 7 Report Generation steps in distributed ALE system

Figure 7 shows a p2p architecture composed by 5 ALE nodes, 4 readers and a business application. The previously presented specification will be send by the business application to one node of the network (here, ALE 1). ALE 1 that receives the specification knows its readers and can know that Reader 1 involved in the ECSpecs is not connected to it, but Reader 2 is. Here comes the hash function, which allows ALE 1 questioning the p2p structure. By hashing the logical reader 1 name Reader1, ALE 1 knows the ALE responsible to store information about this EPC (ALE 5), and can query it (arrow 4). ALE 5 then answers with the URL of ALE 4 connected to the Reader 1 (arrow 5). Then ALE 1 can split the ECSpecs files and sends one to ALE 4 (arrow 6). In this case the splitting operation is easy, we just need to split readers in two ECSpecs file, one for ALE 1 with only Reader2 and one for ALE 4 with only Reader1. The first ECSPecs is kept and used by ALE 1 while the second is sent to ALE 4 via the Web Service URL received from the lookup operation. The two involved ALEs can now process locally with their ECSpecs. It means configuring local reader, launching the inventory, filtering and aggregating reader events, building the report. Once reports are ready, ALE 1 knows that a second report is needed, and waits for it. ALE 4 will send its reports to ALE 1 (arrow 7) which can then merge reports. Finally, ALE 1, the contacted node, sends the report to the business application (arrow 8).

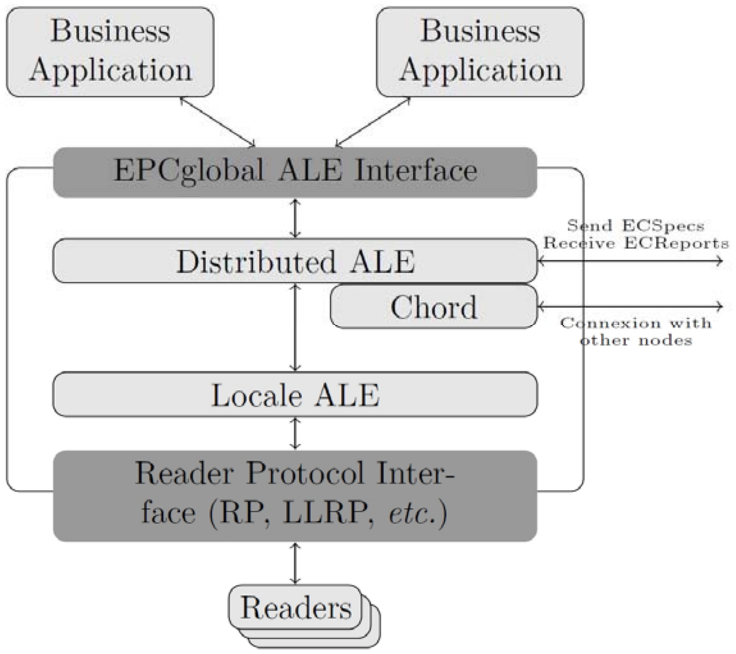


Figure 8 Distributed Filtering and Collection server

Figure 8 presents the architecture of the distributed Filtering and Collection server. The interfaces that are used are compliant with existing EPCGlobal standards, which means that there is no need of re-development for business applications, readers drivers, etc... The distributed ALE component split, send, wait and merge EPCGlobal parts (ECSpecs and ECReports). The Chord protocol is used here for the p2p layer. This is this component that is able to retrieve URL of ALE nodes from a reader name. The local ALE is a typical ALE EPCGlobal compliant.

6.3.2 Current implementation

We currently use the fosstrak 1.0.2 fc-server as locale ALE. The Chord layer is implemented using openChord.

6.4 Embedded Filtering and Collection Server

6.4.1 Design and Principles

The typical use case is illustrated by Figure 9 below:

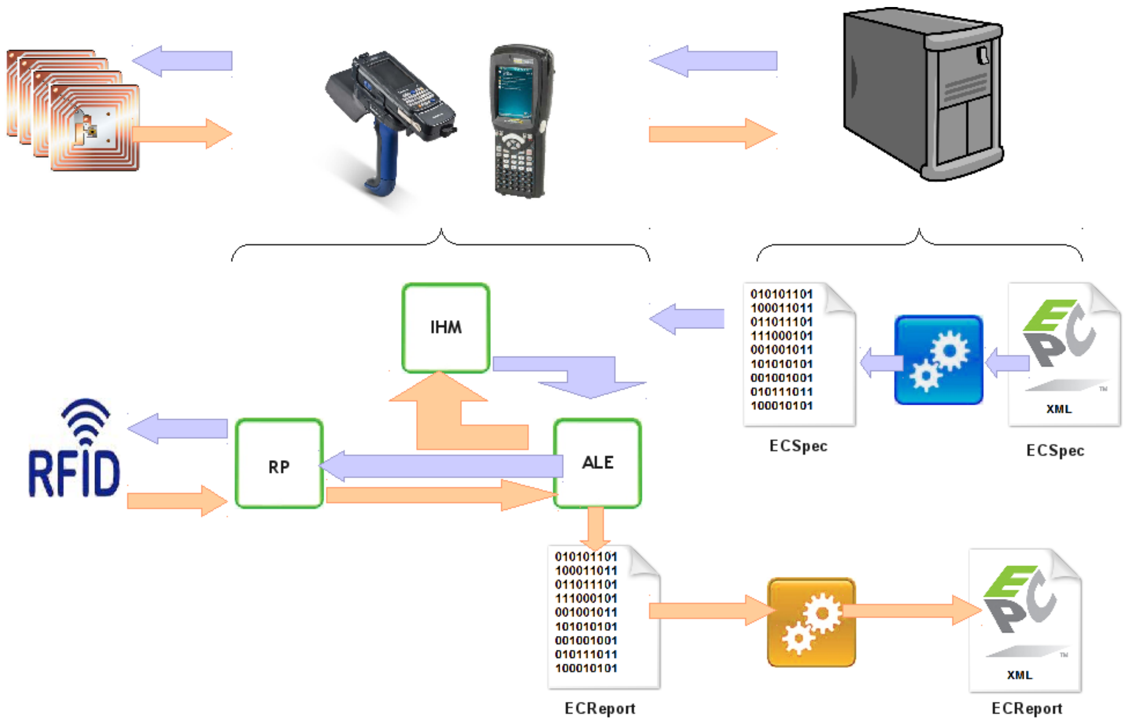


Figure 9 Typical Usage Scenario

There are three main levels in this scenario:

1. **Tags:** EPC1G2 tags sticked on items or location spots.
2. **Embedded applications:**
   - *IHM:* the user's interface for inventory management: start, pause, stop, download ECSpec and upload ECReports in a binary serialized fashion. This application is used as client for the ALE.
   - *ALE:* The filtering and collection server. Tailored for use in an embedded Java environment.
   - *RP:* the reader abstraction layer using the EPC Reader Protocol API. It interacts with the RFID reader device.

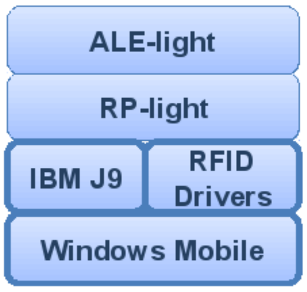


Figure 10 Embedded applications

3. **Server:** connected to an EPC network, it hosts specs and manages reports. The XML marshalling for ECSpec and ECReports objects is performed on the server for performance issues.

The embedded applications are fully written in Java, under the J2ME CDC 1.0 profile. They run on top of IBM's J9 virtual machine. The use of the Java language constrains the development style in order to cope with garbage collection (such as reusable objects).

Finally, the RFID readers (CAEN and BRI) are accessed using vendor-specific drivers.

6.4.2 Current Implementation

As illustrated in Figure 11, the currently implemented layers are the following:

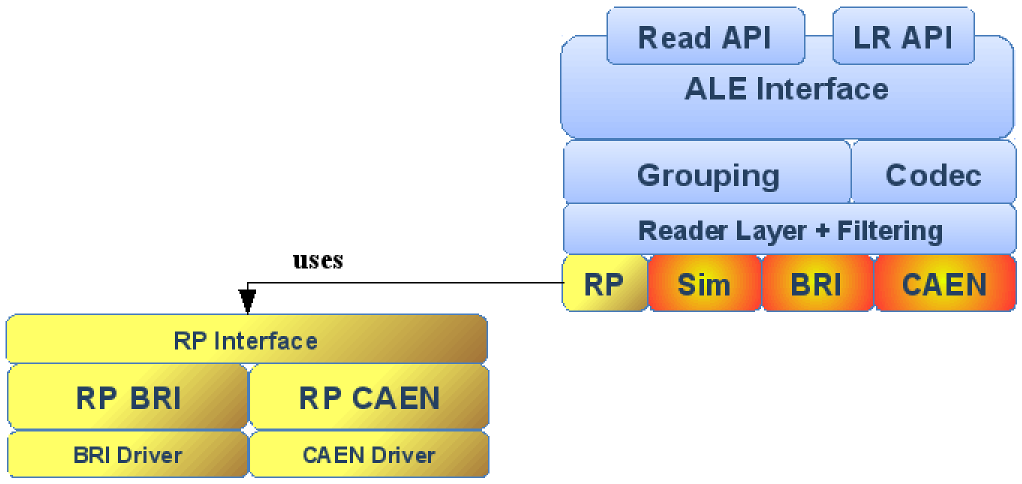


Figure 11 Developed software layers

6.4.2.1 ALE layer

In order to provide minimal inventory services, at interface level, subsets of the Reading and the Logical Reader APIs are implemented:

- Immediate mode: sufficient for user-triggered inventory.
- Fixed readers configuration: only some properties (Power, Session and Initial Q) can be updated.

The ALE engine manages tag grouping according to EPC standard patterns in input ECSpec objects.

A lightweight custom CODEC was developed as well, in order to decode tag IDs using binary format (array of bytes) and in a garbage-free fashion. A filter engine is also made available for software filtering of tags. This leaves the choice for Reader Connectors to choose the best tradeoff between software and hardware filtering.

Note that, because of the Java CDC constraint, the ECSpec and ECReports classes and subclasses were written manually despite of automatic generation from XSD files.

6.4.2.2 Reader Layer

This abstraction layer defines the contract that a reader connector shall respect in order to be bound to the ALE. It provides several services to the connectors such as access to the software filtering engine. Four connectors have been developed:

- *RP*: connector to an RP-compliant reader implementing the minimal inventory functions.

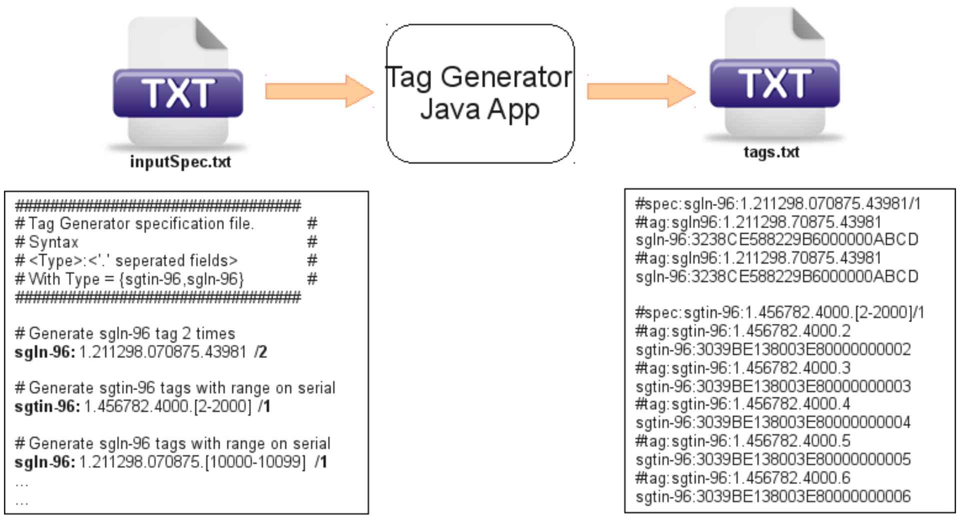
- *Sim*: a custom reader for debugging purposes. The list of tags for simulation can be retrieved from a configuration file generated by the tag generator application (see next section).
- *BRI*: Connector to the Intermec IP30 reader using the BRI driver.
- *CAEN*: Connector to the CAEN A528 reader using the CAEN driver.

6.4.2.3 EPC Reader Protocol wrapper (RP)

This wrapper defines the Reader Protocol interface classes that are used to dialog with an RP-compliant reader device. Based on each vendor-specific driver, two implementations were developed in order to provide minimal required services (inventory).
Note that the communication with the ReaderDevice is done locally and directly via method calls. This avoids overhead when using MTB layers for message bindings.

6.4.3 Other Tools

- Tag Generator: In order to help debugging the ALE engine, the following tag generator application was developed.



```
####################################
# Tag Generator specification file.        #
# Syntax                                   #
# <Type>:<'.' seperated fields>            #
# With Type = {sgtin-96,sgln-96}           #
####################################

# Generate sgln-96 tag 2 times
sgln-96: 1.211298.070875.43981 /2

# Generate sgtin-96 tags with range on serial
sgtin-96: 1.456782.4000.[2-2000] /1

# Generate sgln-96 tags with range on serial
sgln-96: 1.211298.070875.[10000-10099] /1
...
...
```

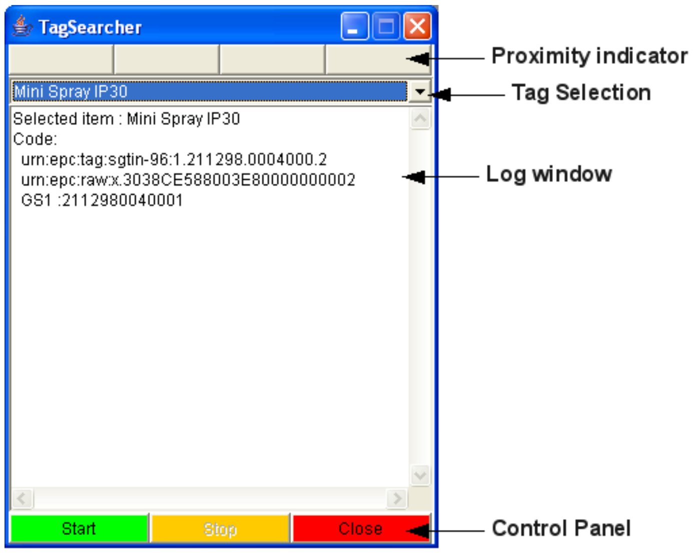
```
#spec:sgln-96:1.211298.070875.43981/1
#tag:sgln96:1.211298.70875.43981
sgln-96:3238CE588229B6000000ABCD
#tag:sgln96:1.211298.70875.43981
sgln-96:3238CE588229B6000000ABCD

#spec:sgtin-96:1.456782.4000.[2-2000]/1
#tag:sgtin-96:1.456782.4000.2
sgtin-96:3039BE138003E80000000002
#tag:sgtin-96:1.456782.4000.3
sgtin-96:3039BE138003E80000000003
#tag:sgtin-96:1.456782.4000.4
sgtin-96:3039BE138003E80000000004
#tag:sgtin-96:1.456782.4000.5
sgtin-96:3039BE138003E80000000005
#tag:sgtin-96:1.456782.4000.6
sgtin-96:3039BE138003E80000000006
```

Figure 12 Tag Generator

Based on a specification input file, a list of tags is generated in binary hexadecimal format. This list of tags can then be used to provide ALE engine a flow of tags in order to tests grouping and filtering patterns. This application is based on the same CODEC used by the ALE engine.

- Tag Searcher: In order to help searching for a tag, this application modulates a beep frequency depending on distance to a tag. The input list of tags (in TAG-ENCODING-URI format) may be local or on a remote HTTP server.



```
Selected item : Mini Spray IP30
Code:
 urn:epc:tag:sgtin-96:1.211298.0004000.2
 urn:epc:raw:x.3038CE588003E80000000002
 GS1 :2112980040001
```

Figure 13 Tag searcher

6.5 OSGi Implementation

6.5.1 Motivation

Actually the ALE server implementation embeds an important number of existing libraries and frameworks in order to provide its functionality. For example, the CXF framework is used to supply SOAP-protocol support to the Web Services implementation of ALE interfaces (expressed in WSDL language) defined by EPCGlobal [8]. CXF it-self is based in other frameworks such as Spring [20], and JAXB.

Moreover, the ALE server implementation uses libraries to supply other required tasks like tag format translation (TDT component), LLRP support (LLRP toolokit [7]), etc. In consequence, deployment of the ALE server on a Web container (e.g. Tomcat server) requires the construction of a unique and large artifact (a war file) that embeds all libraries, and the construction of a monolithic implementation (the ALE-Server implementation it-self).

This monolithic implementation entails evolution problems in two aspects:

- In the ALE Server it-self, when new features needed to be implemented of bugs have to be fixed, the compiled classes (class files) must be added or replaced into the deployed ALE server war file.
- When new version of used libraries are provided by vendors, integration of these new versions (or fixed ones) is an error-prone and complex task. New libraries artifacts (jar files) must be embedded to the ALE server war file as well.

Furthermore, the ALE Server must meet specific extension requirements as addition of new drivers to support different physical RFID readers. With today monolithic implementation, addition of these drivers requires:

- Inclusion of compiled classes that correspond to the reader driver into the war file.
- Inclusion of required libraries used by the driver code into the war file.

- Stopping the ALE server application, and redeployment of the war file in the container (Tomcat)

However in some situations, stopping the server is not a possible choice because some business functions must be active all time.

In order to attack these problems, an implementation of the ALE server on OSGi platform has been created in the Aspire project. This implementation provides to the Aspire RFID middleware important properties such as:

- Evolution: Used libraries and frameworks can be provisioned as modules in order to make substitution easier.
- Modularity: Extensions to the middleware as addition of new readers or sensors should be provided in a modular fashion. Using OSGi approach, new readers are developed without entering their driver code in the same archive (war) used to package the ALE Server.
- Dynamicity: These new modules will be deployed (or undeployed) on the platform dynamically without stopping the applications that are currently in execution.

6.5.2 Architecture on OSGi platform

Implementation of the ALE Server on OSGi platform requires a different architectural style, even if the code source of basic implementation remains almost unmodified. Indeed, in the OSGi architectural style each component of the architecture is deployed as a module (or bundle in OSGi jargon). Each bundle must define explicitly its requirements and capabilities, at runtime the OSGi platform implementation (i.e. Felix [15]) verifies if a bundle is able to be active, that is, if all its requirements are supplied by the platform or by other bundles.

Following these architectural style, the different used libraries have been deployed into the OSGi platform as bundles. The ALE server core is it-self deployed as a bundle on the OSGi platform. For simplicity, some of the ALE server core dependencies were embedded into the core bundle. The objective is provisioning all used libraries as OSGi bundles.

6.5.3 Provisioning of physical reader drivers in OSGi

An important issue of the ALE implementation in Aspire project is the easy integration of new physical reader to the middleware infrastructure. Using the OSGi implementation each new reader driver is implemented as a new bundle. Moreover, the reader driver bundle must define the required bundles for its correct operation, the ALE server is not aware of those dependencies. This property isolates the development of the ALE server core from development of its extensions because modifications have not to be performed in the core bundle. An example of a driver provisioning is presented in Figure 14.

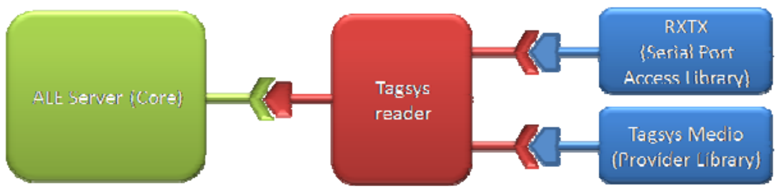

ALE Server (Core)
Tagsys reader
RXTX (Serial Port Access Library)
Tagsys Medio (Provider Library)

Figure 14 Tagsys Reader Driver Provisioninng

Four new reader drivers are provided in this way:

- Tagsys reader driver: it uses a legacy library (Tagsys Medio) and a library to access serial and parallel ports (RXTX). Three bundles are then deployed on the OSGi platform, one for each library and other for the driver code.
- ACR122u reader driver: it uses an open source library in order to access physical device. Two bundles are deployed on the OSGi platform.
- ProximaRF reader driver: it uses a library in order to acess its provided native C library. Two bundles are deployed on OSGi platform.
- Simple reader driver: it's a bundle containing a driver for a fictive reader used for test purposes.

The complete architecture of ALE server core with reader drivers extension is illustrated in Figure 15.

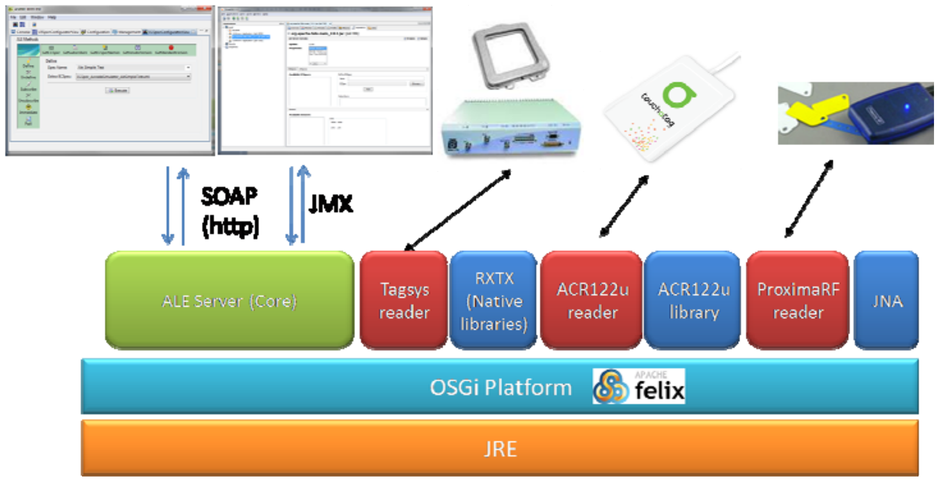


Figure 15 ALE Server Architecture on OSGi Platform

6.6 Sensor Data Inclusion

6.6.1 Motivation

Actually, ALE implementations are processing only in the GUID read from RFID tags, however lots of applications are interested also in tag content. In addition to RFID tag content, sensor information is also relevant for an important number of applications. These sensor-based applications are interested on current physical measurement as temperature, humidity, geo-location, shocks, etc.

For example, the cold chain use case is a type of application that uses information of sensors in order to assure properties in storage and distribution of some kinds of sensible products. These products must remain in a range of temperature in all moment, for example in pharmaceutical industry, some kinds of vaccines must remain between 2°C-8°C. In food industry, some products must not be exposed to temperatures exceeding a threshold for a predefined period of time.

Those use cases require information allowing identification of products (RFID tags), but also require gather information from sensor presents in the environment. Moreover, some types of

RFID tags (semi-passive and active tags) have the capability of maintain historical information (data loggers) of physical measurements. In order to meet these requirements, extensions must be implemented in ALE server to provide the capability of interrogate different sensors and correlate their information with products GUIDs.

Other characteristic of actual implementations of ALE server is the conditions imposed in order to produce reports. Some conditions may interrupt the production of a report, for example when no tags were read in the last event cycle. Differences between the set of tags reported between two successive event cycles may be used also as interruption condition, for example when the same set of tags is read between the two report productions. However, in some cases, applications must be informed only when some conditions are presents. In this case, it is necessary add the capacity of specify conditions to report production in different event cycles specification.

EPC-Global had added a simple extension mechanism in ECSpec format. Indeed, ECSpecs are expressed in XML, and the <extension> XML element may be used to indicate extensions in the ECSpec (Event Cycle Specification). The specification states that syntax and semantic of extensions are responsibility of ALE server implementation. The produced reports have also the <extension> XML element to put data associated to the extensions.

Two kinds of extensions have been added into the ALE server by the Aspire project. The first one allows include sensor data into the reports. The second permits adding conditions the report production. The next subsections present each type of extension.

6.6.2 Inclusion of sensor information

The ECSpec format has been extended to ask the ALE server information from deployed sensor into the environment. The client indicates declaratively the information to be included and conditions in order to identify the source of the information.

A list of required information is indicate into the <asp:inclusions> XML element. For each required information, a query expressed as an *LDAP query* is used. Each expression must be into a <asp:inclusion> XML element.

In Figure 16 is illustrated the extension format, in this example there are two inclusion statements. The first one states the inclusion of ***temperature*** data from sensors which ***location*** is ***indoor***. In the second one, the ***pressure*** data is gathered from sensors which ***location*** is ***outdoor***.

```
<reportSpec reportName="MySimpleReport">
   <reportSet set="CURRENT" />
   <output includeRawHex="true" includeRawDecimal="true"
      includeEPC="true" includeTag="true" includeCount="true" />
   <extension>
      <extension>
         <asp:aspire-extension
            xmlns:asp="http://aspirerfid.ow2.org/ECSpec/extensions"

xsi:schemaLocation="http://aspirerfid.ow2.org/ECSpec/extensions
ECSpecPP.xsd ">
```

```
            <asp:inclusions>

<asp:inclusion><![CDATA[(&(data.type=temperature)(sensor.location=ou
tdoor))]]></asp:inclusion>

<asp:inclusion><![CDATA[(&(data.type=pressure)(sensor.location=indoo
r))]]></asp:inclusion>
            </asp:inclusions>
         </asp:aspire-extension>
      </extension>
   </extension>
</reportSpec>
```

Figure 16 Inclusion of sensor data into ECSpec

At runtime, the extended implementation of ALE server interprets the ECSpec with additional requirements and includes information from sensors meeting the conditions expressed in the query. If no conditions were expressed, information for every sensor producing the adequate measurement is included in the environment.

6.6.3 Condition on report productions

Condition extensions allow added conditions to report production. In each event cycle, the conditions are evaluated, if the result of evaluation is positive the ECReport is then produced, otherwise it is not.

In order to express conditions on reports production, the <asp:conditions> XML element is used. This element has the property ***type*** that may have two values ***all*** or ***any***. If ***all*** is indicated all conditions must be evaluated to true to produce the report. On the other hand if ***any*** is indicated, when one condition into the set is evaluated to true the report is produced. Each condition is expressed using and LDAP query expression and it is added into a <asp:condition> XML element.

In Table 6 is illustrated an example of a condition set on the production of the report ***OtherReport***. In this example, the attribute ***type*** has the value ***any***, indicating that the two conditions must be true to allow the report production by the ALE server. The first condition states that ***altitude*** property must be ***inferior*** to ***2000*** mt. The second one states ***temperature*** property must be superior to ***273*** degrees Kelvin.

At runtime, the ALE server interprets expressions and determines if the report must be produced.

```

<reportSpec reportName="OtherReport">
   <reportSet set="CURRENT" />
   <output includeRawHex="true" includeRawDecimal="true"
       includeEPC="true" includeTag="true" includeCount="true" />
   <extension>
      <extension>
         <asp:aspire-extension
            xmlns:asp="http://aspirerfid.ow2.org/ECSpec/extensions"
```

```
xsi:schemaLocation="http://aspirerfid.ow2.org/ECSpec/extensions
ECSpecPP.xsd ">
           <asp:conditions type="All">

<asp:condition><![CDATA[(altitude<2000mt)]]></asp:condition>

<asp:condition><![CDATA[(temperature>273K)]]></asp:condition>
           </asp:conditions>
        </asp:aspire-extension>
     </extension>
   </extension>
</reportSpec>
```

Table 6 Inclusion of report production condition into ECSpec

6.6.4 Extension Architecture

The OSGi ALE server architecture presented in previous section of this document leverages the addition of new software components to ALE server in a modular and dynamic fashion. In addition, the open source ***Wire Admin Binder*** (WAB) framework allows easily implementation of sensor-based applications. The WAB framework is provided by two OSGi bundles that have been deployed into the OSGi platform containing the ALE server.

Moreover, each of different sensor presents on the environment has to implement a driver and provisioning a service into the OSGi platform (OSGi is a SOA platform). These services are then used by the WAB framework to poll data from sensors. The architecture of extensions is illustrated by Figure 17.

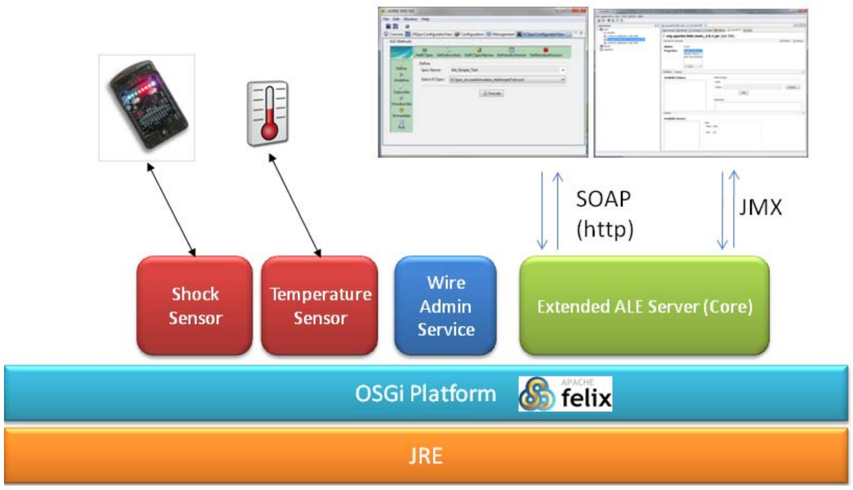


Figure 17 Architecture of the extended ALE server

Section 7 Business Event Generator

7.1 Overview and purpose

The architecture introduces a Business Event Generator (BEG) component between the F&C and Information Sharing (e.g., EPC-IS) components as shown in Figure 18 below. The role of the BEG is to automate the mapping between reports stemming from F&C and IS events. Instead of requiring developers to implement the mapping logic, the BEG enables application builders to configure the mapping based on the semantics of the RFID application.

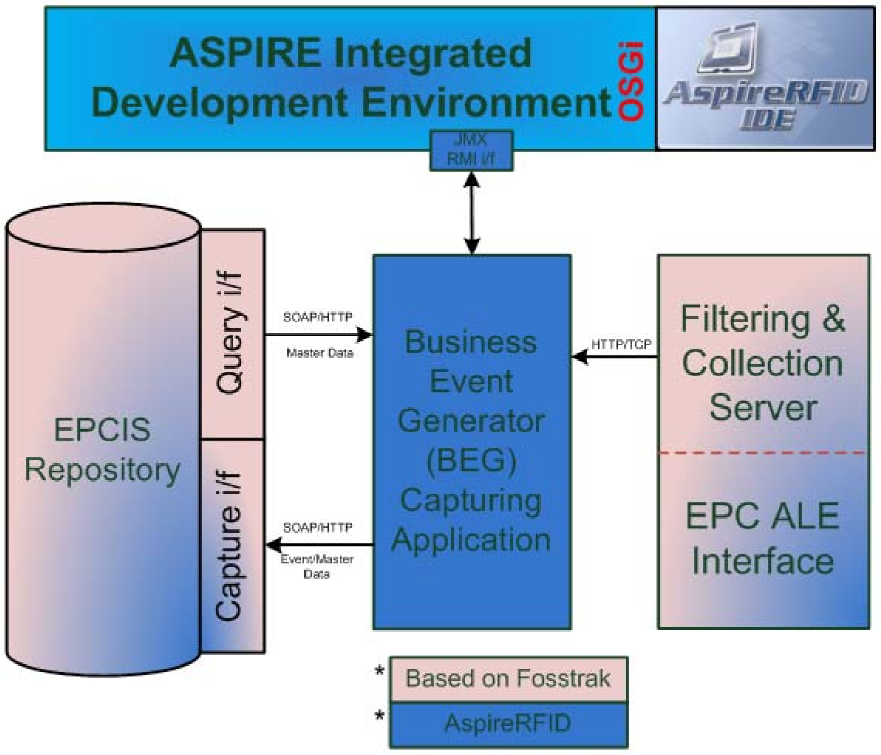


Figure 18 BEG Interfaces

With the help of AspireRfid IDE by describing the company's business processes and its underlying business infrastructure, the required business events that constitute a company's business functionality are created and stored at the RFID repository that Business Event Generator engine exploits to define its functionality. In EPC terms, BEG can be seen as a specific instance of an EPC-IS capturing application, which parses EPC-ALE reports, fuses these reports with business context data using the assigned business event from the company's business metadata to serve as guide and accordingly prepare EPC-IS compliant events. These events are submitted to the EPC-IS Repository, based on an EPC-IS capture interface and related bindings (e.g., HTTP/JMS). The specification of the BEG is a valuable addition over existing RFID middleware architectures and platforms.

In terms of the EPCglobal Architecture, the BEG is an intermediate layer between the ALE and the EPCIS, as depicted in Figure 19 below.

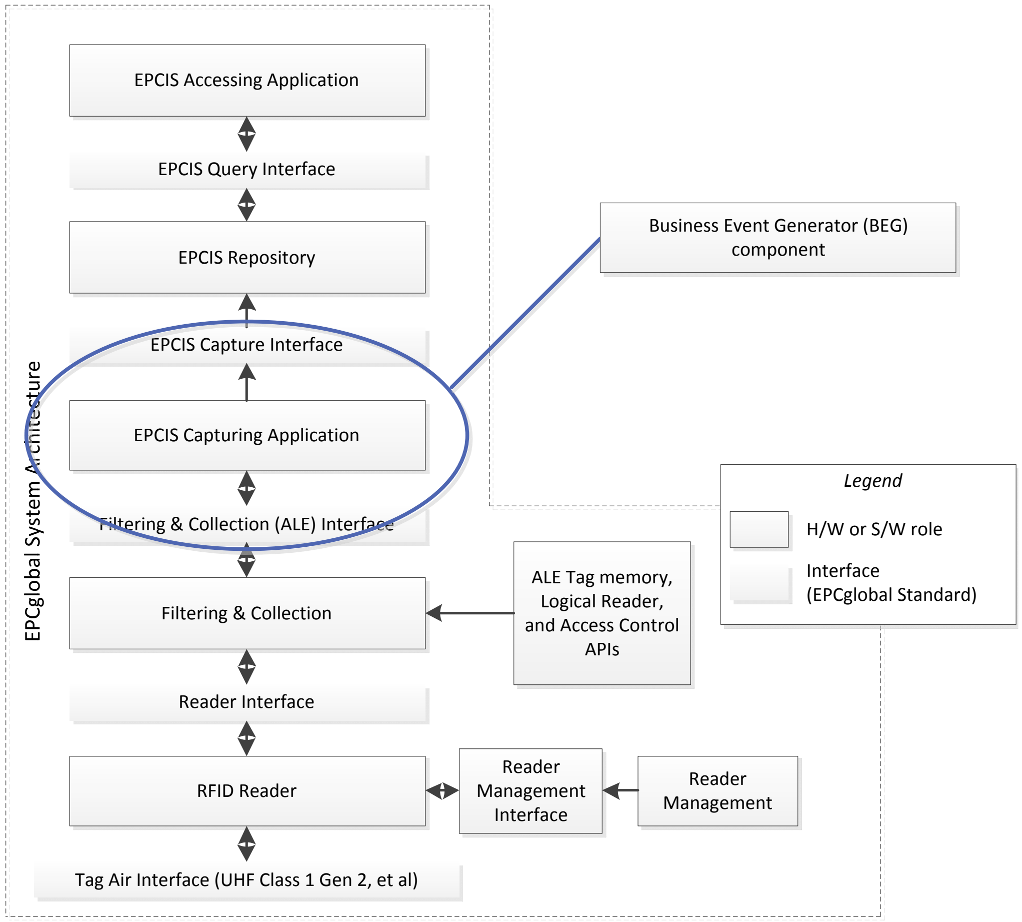

EPCIS Accessing Application
EPCIS Query Interface
EPCIS Repository
EPCIS Capture Interface
EPCIS Capturing Application
Filtering & Collection (ALE) Interface
Filtering & Collection
Reader Interface
RFID Reader
Tag Air Interface (UHF Class 1 Gen 2, et al)
EPCglobal System Architecture
Business Event Generator (BEG) component
ALE Tag memory, Logical Reader, and Access Control APIs
Reader Management Interface
Reader Management
Legend
H/W or S/W role
Interface (EPCglobal Standard)

Figure 19 Position of the BEG component according to the EPCglobal architecture

According to EPCglobal, the BEG component is the EPCIS Capturing Application required to operate between the F&C and EPCIS layers of the logical architecture

7.2 Interfaces to other components

The BEG component interface provides five methods for interaction with the BEG client which are all communicating with the BEG client by exchanging SOAP messages.

1. The first method is the `getEpcListForEvent (EventStatus getEpcListForEvent(String eventID))` which is used for returning to the BEG client an `EventStatus` object which contains the real time list of EPC ids and the transaction ID of a chosen Event (String eventID) from the list of events that the BEG component is already serving. So with the help of this method one can observe at real time the incoming IDs as they are reported to the BEG by the F&C component and are related with a specific transaction Event.
2. The second method is the `stopBegForEvent (boolean stopBegForEvent(String eventID))` which is used by the BEG client to stop serving a predefined `Event` by sending to it its specific `EventID`.

3. The third method is the `getStartedEvents` (`List<String> getStartedEvents()`) which returns a list of Event IDs that the BEG component is serving.
4. The fourth method is the `startBegForEvent` (`boolean startBegForEvent(VocabularyElementType vocabularyElementType, String repositoryCaptureURL, String begListeningPort)`) which is used to set up the BEG component for start serving a specific Event. More specifically this method takes the already pre described Elementary Business Transaction Event described at the Information Sharing repository's Master Data and uses it for configuring the Business Event Generator to create Business Events from the ECReports received from the port given as variable to the `startBegForEvent` method. If the method is successful it will return `true` otherwise it will return `false`.
5. Finally, the fifth method is the `getEventList` (`List<VocabularyElementType> getEventList( String repositoryQueryURL)`) which is used for returning a list of all the available defined Events from a Company's EPCIS Master Data repository.

7.3 Implementation details

Technically, the BEG component is packed as a .war file which is deployed on top of an Apache Tomcat 6.0 container (or higher) and uses Java version 1.6 (and higher).

In order to implement the Web Services required for its configuration and management, the frameworks JaxWS and CXF/Spring were used.

For capturing the produced ECReports from the Filtering and Collection layer it provides a TCP/HTTP interface that is configured by defining different capturing port for each Elementary Business Transaction's defined report ID.

It uses Fosstrak's Event Data Capture client implementation which follows the EPCIS 1.1 Capture interface which uses the JaxWS framework and CXF/Spring technologies for implementing the required Web Services and the "construction" of the SOAP messages.

It uses Fosstrak's Master Data Query client implementation which follows the EPCIS 1.1 query interface which uses the JaxWS framework and CXF/Spring technologies for implementing the required Web Services and the "construction" of the SOAP messages

And finally the Event Types that the Business Event Generation component supports are the standard Event types defined in the EPCIS 1.1 Specifications which are:

- Quantity Events,
- Aggregation Events,
- Transaction Events and
- Object Events

7.4 Configuration example

The objective of this tool is to provide a control client to configure Business Event Generator (BEG) so as to translate ECReports to specific EPCIS Events by taking in consideration the already defined Master Data. Moreover, it provides a view which is able to show the real time processed readings for each event served at that time. A screenshot of the BEG plugin is presented in Figure 20 below.

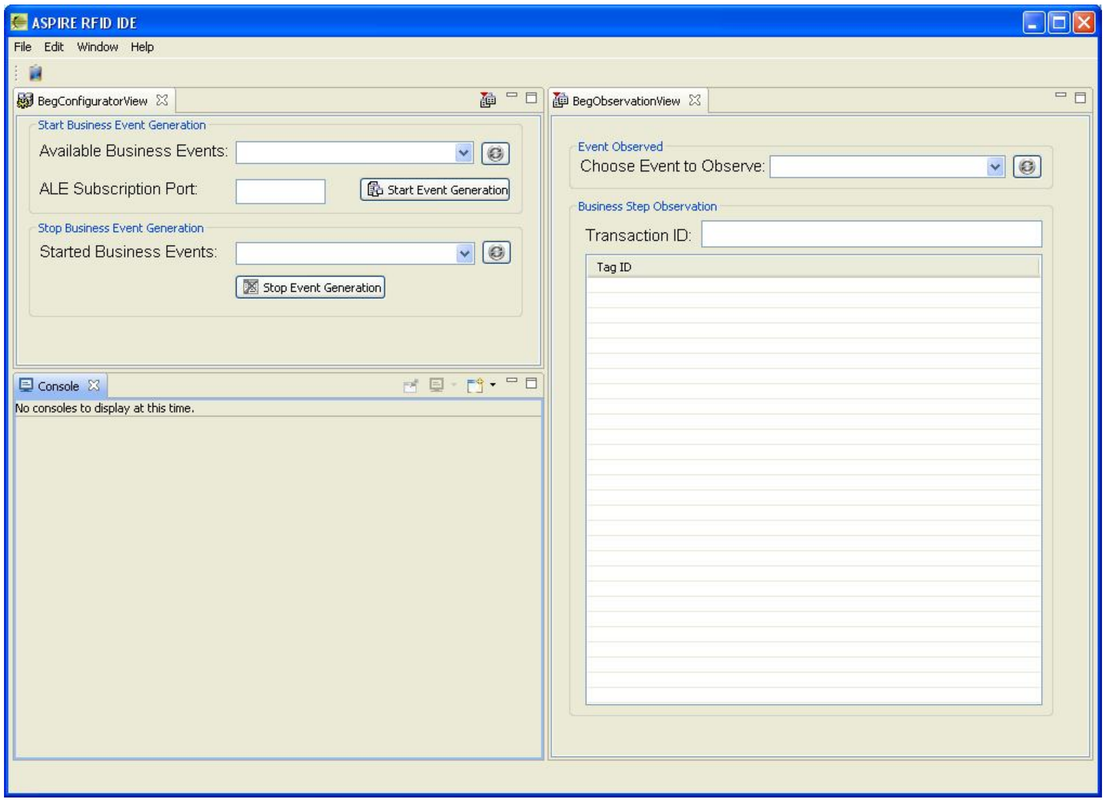


Figure 20 BEG plugin screenshot

In order to configure the BEG plugin, the following need to be defined:

- EPCIS Repository Capture Endpoint: Where the connections for the Capturing Interface are being accepted.
- EPCIS Repository Query Endpoint: Where the connections for the Query interface are being accepted.
- Event/Port Binding: Predefine event/port bindings.
- Observations Refresh Rate: Specify how often the observation server will update its view.

These can be defined through Window -> Preferences -> BEG in the ASPIRE IDE (http://wiki.aspire.ow2.org/xwiki/bin/view/Main.Documentation.AspireIDE/BusinessEventGeneratorPlug-in).

Section 8 EPCIS

8.1 Overview and purpose

The ASPIRE Information Sharing repository and services are based on the EPCIS specification [10] and are the components that:

- Receive application-agnostic RFID data from the F&C server through the Business Event Generation (BEG) component.
- Translate RFID data in corresponding business events. These events carry the business context as well (i.e., they refer to particular companies, business locations, business processes etc.).
- Make business events available and accessible to other upstream applications.

The ASPIRE Information Sharing component consists of three parts:

- A capture application that interprets the captured RFID data,
- A repository (i.e. a database system) that provides persistence, and
- A query application that retrieves the business events from the repository.

Note that the ASPIRE Information Sharing repository:

- Deals explicitly with historical data (in addition to current data).
- Deals not just with raw RFID data observations, but also with the business context associated with these data (e.g., the physical world and specific business steps in operational or analytical business processes).
- Operates within enterprise IT environments at a level that is much more diverse and multi-faceted comparing to the underlying data capture and Filtering & Collection middleware components.

Generally, the ASPIRE information sharing repository is built to deal with two kinds of data:

- RFID event data i.e. data arising in the course of carrying out business processes. These data change very frequently, at the time scales where business processes are carried out.
- Master/company data, i.e. additional data that provide the necessary context for interpreting the event data. These are data associated with the company, its business locations, its read points, as well as with the business steps comprising the business processes that this company carries out.

8.2 Current implementation status and Interfaces to other components

Business events are generated at the edge and delivered into the Information Sharing middleware infrastructure through an appropriate capture interface as shown in Figure 21. The BEG middleware undertakes to automatically map application agnostic reading (from the F&C layer) to the Information Sharing middleware. These events can be subsequently delivered to interested enterprise applications through the interface enabling query of RFID business events.

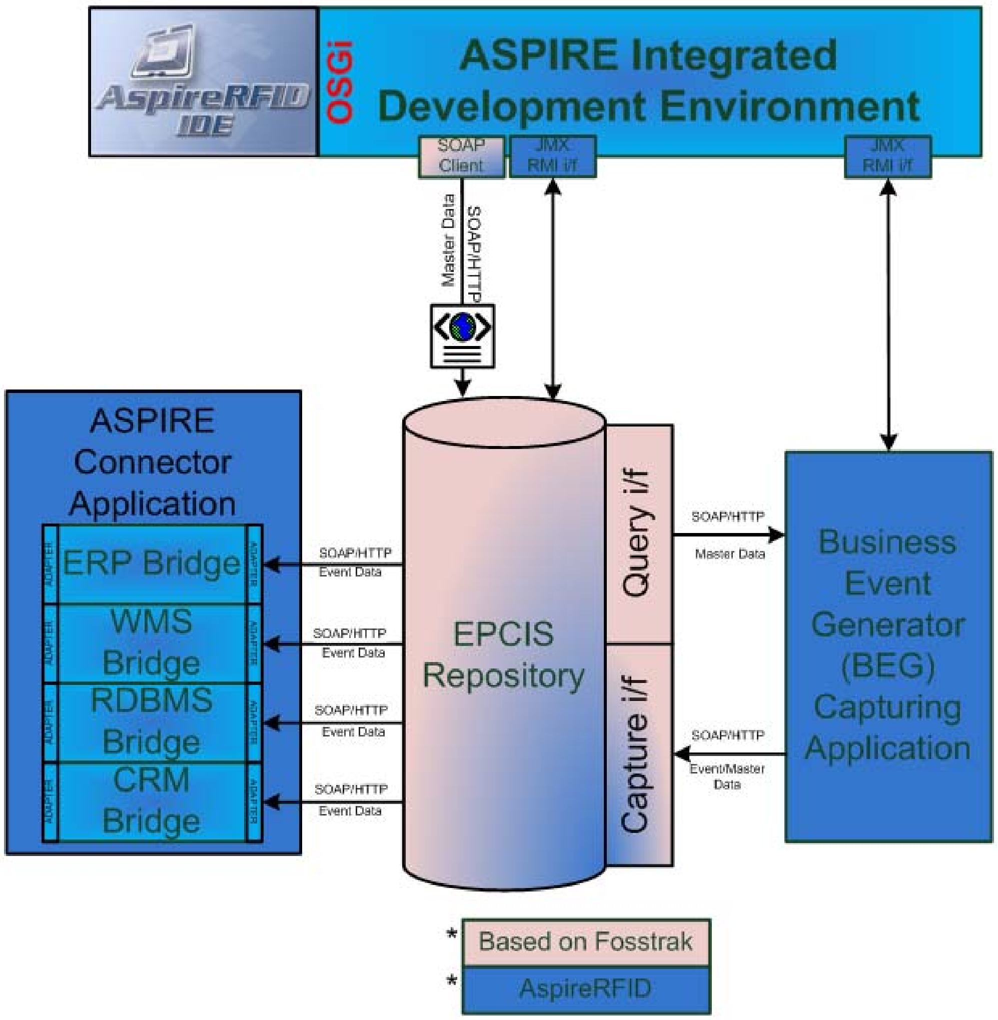


Figure 21 EPCIS Interfaces

Note that the AspireRfid EPCIS Repository is a modified version of Accada RFID Middleware, which is now called Fosstrak, licensed under LGPL. AspireRfid has implemented changes and bug fixes on the original Accada middleware. These changes and bug fixes are summarized in Table 7 below.

S/N	Description/Modules and Rationale
	Enhancements:
1	ASPIRE have created a Master data (Web Service Based) Capture API (Client/Server implementation) that supports: • Alter, a vocabulary's Element URI. o simpleMasterDataAlter(String vocabularyType, String oldVocabularyElementURI, String newVocabularyElementURI) • Insert, a vocabulary's Element . o simpleMasterDataInsert(String vocabularyType, String vocabularyElementURI) • Insert, many vocabulary's Elements .

- simpleMasterDataMassInsert(String vocabularyType, ArrayList<String> vocabularyElementURIs)

- Delete a vocabulary's Element (When using simple delete only the element with its attributes can be deleted).
  - simpleMasterDataSingleDelete(String vocabularyType, String vocabularyElementURI)
- Delete many vocabulary's Elements (When using simple mass delete only the list of elements with their attributes will be deleted).
  - simpleMasterDataMassDelete(String vocabularyType, ArrayList<String> vocabularyElementURIs)
- Delete a vocabulary's Element with its direct or indirect descendants (The element with its attributes and with all its children elements and its children's attributes).
  - simpleMasterDataDeleteWithDescendants(String vocabularyType, String vocabularyElementURI)
- Delete many vocabulary's with their Elements and with their direct or indirect descendants. The elements with its attributes and with all its children elements and its children's attributes will be deleted.
  - simpleMasterDataMassDeleteWithDescendants(String vocabularyType, ArrayList<String> vocabularyElementURIs)
- Mass Insert or Update a vocabulary's Element Attributes. If the Vocabulary is not inserted yet it will be inserted. The vocabularyURI, vocabularyAttribute pair should be unique so if it already exists it will be changed to the vocabularyAttribute entered or simply rewrite it.
  - simpleMasterDataAndMassAttributeInsertOrAlter(String vocabularyType, String vocabularyURI, HashMap<String, String> vocAttributes)
- Insert or Update a vocabulary's Element Attribute. If the Vocabulary is not inserted yet it will be inserted. The vocabularyURI, vocabularyAttribute pair should be unique so if it already exists it will be changed to the vocabularyAttribute entered or simply rewrite it.
  - simpleMasterDataAndAttributeInsertOrAlter(String vocabularyType, String vocabularyURI, String vocabularyAttribute, String vocabularyAttributeValue)
- Delete many vocabulary's Element Attributes.
  - simpleMasterDataAttributeMassDelete(String vocabularyType, String vocabularyURI, ArrayList<String> vocabularyAttributeIDs)
- Delete a vocabulary's Element Attribute.
  - simpleMasterDataAttributeDelete(String vocabularyType, String vocabularyURI, String vocabularyAttribute)

Each of the above methods produce a single EPCIS MasterData document which is send via web services to the EPCIS's current capture interface where it is handled appropriately. The database operations are using Hibernate following the current Fosstrak Capture interface implementation.
The Reason for creating such an interface is that whereas EPC defines a capture API for RFID Events it doesn't define one for Master Data which is required for applications, if we do not want to give direct control over the EPCIS's database, (e.g. AspireRFID Master Data Editor http://wiki.aspire.ow2.org/xwiki/bin/view/Main.Documentation.AspireIDE/MasterDataEditor) to store Master Data to the EPCIS repository.

2	ASPIRE have changed the way Vocabulary's children are stored to the database which were divided with the "_" sign to the "," sign (so as not to create faulty trees if someone uses the "_" sign in a URI description).
3	ASPIRE have updated Apache CXF library for the web service implementation from 2.0.4-incubator to 2.2.4
4	Use of the "Universal" common library instead of epcis-commons

Table 7 Bugfixes and additions in the Fosstrak EPCIS implementation

8.3 EPCIS Capture Interface

8.3.1 Master Data Capture Interface

A new feature that ASPIRE architecture introduces is the Master Data Capture Interface as shown in Figure 21 which enables the EPCIS component to capture Master Data in the form of Master Data xml-compliant documents through this specialized interface. By providing this interface a client is capable of adding new, deleting or altering EPCIS's vocabulary data by exchanging SOAP messages. The two client methods that have been created for achieving what is described above are the "simpleMasterDataEdit" and the "simpleMasterDataAndAttributeEdit".

The "simpleMasterDataEdit" method (boolean simpleMasterDataEdit(String vocabularyType, String vocabularyElementURI, String mode)) is used to insert, update or Delete a vocabulary's Element. It takes three variables as input:

- the Vocabulary type
- the Vocabulary element URI
- the mode that the method is "operating".

When using delete mode either only the element with its attributes can be deleted or the element with its attributes and with all its children elements and its children's attributes. When using insert mode if the Vocabulary element is not inserted yet it will be inserted. For using the alter URI mode (2) the Old URI with the new one should be given at the "vocabularyElementURI" parameter merged together with the "#" sign between them (e.g. "urn:epcglobal:old#urn:epcglobal:new"). If the execution of the method is Successful it will return "true" otherwise "false". The supported operational modes are 4:

- mode 1: insert
- mode 2: alterURI
- mode 3: singleDelete
- mode 4: Delete element with its direct or indirect descendants

The "simpleMasterDataAndAttributeEdit" (simpleMasterDataAndAttributeEdit (String vocabularyType, String vocabularyURI, String vocabularyAttribute, String vocabularyAttributeValue, String mode)) can insert, update, delete a Vocabulary's Element Attribute. If the Vocabulary is not inserted yet it will be inserted. The vocabularyURI, vocabularyAttribute pair should be unique so if it already exists it will do nothing except if the mode is set to "2" which alters the chosen attribute. It takes five variables as input:

- the Vocabulary Type
- the Vocabulary URI
- the Vocabulary Attribute

- the Vocabulary Attribute Value
- the methods “operational” mode

If the execution of the method is Successful it will return “true” otherwise “false”. The supported operational modes are 3:

- mode 1: Insert
- mode 2: Alter Attribute Value
- mode 3: Delete Attribute

Section 9 Object Name Service

9.1 Overview and purpose

The Object Name Service (ONS) is a service that returns a list of network accessible service endpoints that pertain to a requested Electronic Product Code (EPC). The ONS does not contain actual data about the EPC; it only contains the network address of services that contain the actual data. The ONS uses the Internet's existing Domain Name System (DNS) for resolving requests about an EPC. In order to use DNS to find information about an item, the item's EPC must be converted into a format that DNS can understand, which is the typical, "dot" delimited, left to right form of all domain-names. The ONS resolution process requires that the EPC being asked about is in its pure identity URI form as defined by the EPCglobal Tag Data Standard [11]. This URI conversion is done in the local server by the TDT component.

The key components of the logical architecture are the Core ONS servers. The core of the system comprises by the ONS servers themselves. Each company assigned part of the global EPC namespace is responsible to create and maintain a functional ONS server to serve queries about EPC's inside that namespace. Each ONS server is basically a standard DNS server with special NAPTR records [11], mapping EPC's to service access points – typically EPCIS query and capture interfaces where additional information can be obtained about the EPC in question

In the scope of the projects, these servers have been implemented using the BIND open source software (http://www.isc.org/software/bind), upon a Windows OS platform. BIND implementations are also available on all Unix/Linux platforms also. Configuration of these servers remains the same across all platforms. ONS delegation and hierarchy follows the same structure as the DNS hierarchy. Each entity responsible for a specific level in the hierarchy is responsible to maintain the ONS server responsible and also forward requests outside its domain to the appropriate authoritative ONS servers.

Figure 22 represents the logical ONS Architecture. When the manufacturer tags the product, the EPC information related to the product (e.g. manufacture date, location, expiration date, etc.) is stored in its local EPC IS. The product is then shipped to the retailer. Once received, the retailer record some information related to the product in its EPC IS. If the retailer wants to retrieve some manufacturer information related to a specific product, the retailer needs to send a request to the root ONS, which knows the location of the local ONS of the manufacturer (1). The retailer can then send a request to the local ONS of the manufacturer (2), which knows the location of the EPC IS related to the EPC of the specific product requested. Finally, the retailer can access the EPC IS of the manufacturer to retrieve the information related to the product (3).

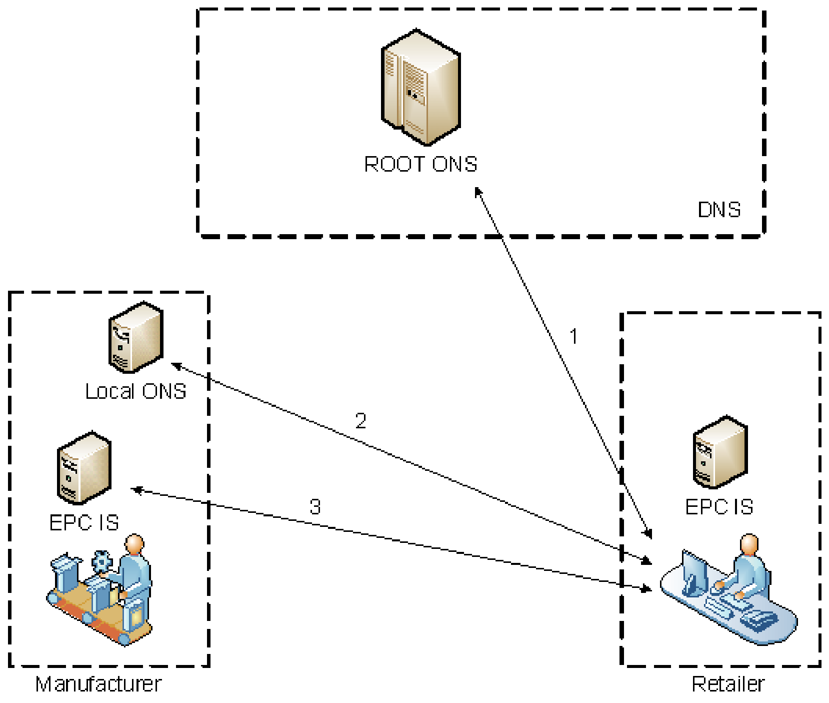


Figure 22 ONS Architecture

If there are several links (e.g. shipping company, large retailer, etc.) in the distribution chain, the ONS will always and only redirect the request to the manufacturer that produced the tag. The data related to a specific EPC and stored in the local EPC IS of the different links cannot be retrieved by requesting the ONS.

9.2 Interfaces to other components

The ONS component is intended to receive inputs from the ONS resolver (translation of the EPC into PURE-ENCODING URI and ONS HOSTNAME), which is part of the Tag Data Translation component (Section 4). It communicates its results to the Business Event Generator and the EPC IS repository, where the information related to the product is recorded.

DNS (and subsequently ONS) servers - as per the standard - can be queried upon using either UDP or TCP requests on port 53. This functionality in our implementation is provided by a small set of Java classes, using the open source dnsjava library (http://www.dnsjava.org/).

9.3 Applications

9.3.1 Demo web application

This application demonstrates the ability of our system to track and present information about the geographical location of a specific EPC id over time. The application provides a google-like search interface, where the user can search for a specific EPC id and plot all available information on a map.

The following list contains the implementation details for our demo application:

- 3-tiered Java web application
- Based on the Struts 2 MVC framework
- Presentation tier (View) uses JSP technology
- Application tier (Controller) uses servlets
- Data access tier (Model) is bound to the EPCIS query interface (XML web services) to retrieve information about a specific EPC id
- Application tier uses the ONS accessing infrastructure to locate the EPCIS repository service access point
- Presentation tier uses the 'Google maps' API to plot information on a geographical map

9.3.2 EPC Id tracking infrastructure.

This application will provide the infrastructure so business partners across a supply chain can bridge their EPCIS repositories, thus enabling them to track EPC Id's as they move across different administrative domains. Our application will run in parallel with the EPCIS repository, as a standalone add-on that can be used when and if necessary. Implementation details are provided in the following list:

- 3-tiered Java web application
- Based on the Struts 2 MVC framework
- Presentation tier (View) uses JSP technology
- Application tier (Controller) uses servlets
- Data access tier (Model) is bound to the EPCIS query interface (XML web services) to retrieve information about a specific EPC id
- Our application also uses a small local database (MySQL implementation)
- Application tier uses the ONS accessing infrastructure to locate the EPCIS repository service access point

The provided functionality is summarized in the following list:

- Select retrieve and view events about any EPC Id contained in the local EPCIS repository. Selection can be filtered based on epcid, time, event_type, epc class, location etc.
- Select and view events referring to 'foreign' EPC Id's. Foreign EPC Id's are considered those belonging to a namespace with a different company identifier (or in general a namespace for which the owner of the system has no authority). Filtering can be done as above
- For foreign EPC Id's, locate the service access point for the information system responsible for them (through ONS)
- For foreign EPC Id's with a valid service access point associated with them, report their arrival within the local administrative domain, to the foreign information system (report is done to the same application running on the remote system , through a web service interface)
- Accept reports from foreign systems for EPC Id's belonging to our domain of authority, and subscribe on behalf of our local EPCIS repository for updates concerning said EPC Id's (again interface is through web services).

- Keep a local record of already reported EPC Id’s (or classes of EPC Id’s) to avoid duplicate subscriptions.

Section 10 Business Intelligence

10.1 Overview and purpose

Business Intelligence (BI) refers to computer-based techniques used in spotting, digging-out, and analyzing business data, such as sales revenue by products or departments or associated costs and incomes.

BI technologies provide historical, current, and predictive views of business operations. Common functions of Business Intelligence technologies are reporting, online analytical processing, analytics, data mining, business performance management, benchmarking, text mining, and predictive analytics.

Business Intelligence often aims to support better business decision-making. Thus a BI system can be called a decision support system (DSS). Though the term business intelligence is often used as a synonym for competitive intelligence, because they both support decision making, BI uses technologies, processes, and applications to analyze mostly internal, structured data and business processes while competitive intelligence, is done by gathering, analyzing and disseminating information with or without support from technology and applications, and focuses on all-source information and data (unstructured or structured), mostly external to, but also internal to a company, to support decision making

As RFID become more and more integrated to modern businesses the amount of data that are automatically handled are exponentially increasing. Querying a large database or a web service that exposes the database becomes very cumbersome and not user friendly. The tool designed and implemented in the scope of the project (AspireBI) aims to be powerful, flexible, and easy to use, web reporting solution that provides browser based, parameter driven, dynamic report generation. This tool aims primarily at delivering a reporting environment over an RFID network.

On top of RFID programmability, the ASPIRE RFID middleware platform was designed to incorporate business intelligence, enabling context analysis over numerous sensors observations. This enables the ASPIRE middleware to record and monitor an RFID network through AspireBI. The tool provides a web based report generation and administration interface with the following features and goals:

- Deliver pixel perfect reports over the EPCIS query framework.
- Will help users to make faster and better business decisions
- Support for a wide variety of export formats including PDF, HTML, CSV, XLS, RTF, and PPT.
- Parameterized reports
- Single entry point for report generation via the web and thin clients, making unnecessary to install client software on each pc.
- Easier to maintain. You create once a report and publish it and all authorized users can see it.
- Everybody see the same thing because there is only one version for each report and not multiple versions for each user
- Persisting data source through a web service connection to the database
- Persisting reports with xml
- It is based upon eclipse BIRT (an open source reporting framework)

10.2 Implementation Description

AspireBI is based on BIRT (Business Intelligence and Reporting Tools), an open source software project that provides reporting and business intelligence capabilities for rich client and web applications, especially those based on Java and Java EE. BIRT is a top level software project within the Eclipse Foundation, an independent not-for-profit consortium of software industry vendors and an open source community.

The project's stated goals are to address a wide range of reporting needs within a typical application, ranging from operational or enterprise reporting to multi-dimensional online analytical processing (OLAP). Initially, the project has focused on and delivered capabilities that allow application developers to easily design and integrate reports into applications.

BIRT has two main components: a visual report designer within the Eclipse IDE for creating BIRT Reports, and a runtime component for generating reports that can be deployed to any Java environment. The BIRT project also includes a charting engine that is both fully integrated into the report designer and can be used standalone to integrate charts into an application.

The reports are extracted from the EPCIS repository and through the EPCIS web service. They are persisted as XML and can access a number of different data sources including SQL databases, JDO datastores, JFire Scripting Objects, POJOs, XML and Web Services.

A rich variety of reports can be added to an application built on top of the ASPIRE middleware, including:

- Lists - The simplest reports are lists of data. As the lists get longer, you can add grouping to organize related data together (orders grouped by customer, products grouped by supplier). If your data is numeric, you can easily add totals, averages and other summaries.
- Charts - Numeric data is much easier to understand when presented as a chart. BIRT provides pie charts, line & bar charts and many more. BIRT charts can be rendered in SVG and support events to allow user interaction.
- Crosstabs - Crosstabs (also called a cross-tabulation or matrix) shows data in two dimensions: sales per quarter or hits per web page.
- Letters & Documents - Notices, form letters, and other textual documents are easy to create with BIRT. Documents can include text, formatting, lists, charts and more.
- Compound Reports - Many reports need to combine the above into a single document. For example, a customer statement may list the information for the customer, provide text about current promotions, and provide side-by-side lists of payments and charges. A financial report may include disclaimers, charts, tables all with extensive formatting that matches corporate color schemes.

Nevertheless, the user has the ability of creating custom reports, thus taming the tool’s powerful capabilities to fulfill any application’s needs. Each report consists of four main parts: data, data transforms, business logic and presentation.

- Data. Databases, web services, Java objects all can supply data to your BIRT report. BIRT provides JDBC, XML, Web Services, and Flat File support, as well as support for using code to get at other sources of data. BIRT's use of the Open Data Access (ODA) framework allows anyone to build new UI and runtime support for any kind of tabular data. Further, a single report can include data from any number of data sources. BIRT also supplies a feature that allows disparate data sources to be combined using inner and outer joins.

- Data Transforms. Reports present data sorted, summarized, filtered and grouped to fit the user's needs. While databases can do some of this work, BIRT must do it for "simple" data sources such as flat files or Java objects. BIRT allows sophisticated operations such as grouping on sums, percentages of overall totals and more.
- Business Logic. Real-world data is seldom structured exactly as you'd like for a report. Many reports require business-specific logic to convert raw data into information useful for the user. If the logic is just for the report, you can script it using BIRT's JavaScript support. If your application already contains the logic, you can call into your existing Java code.
- Presentation. Once the data is ready, you have a wide range of options for presenting it to the user. Tables, charts, text and more. A single data set can appear in multiple ways, and a single report can present data from multiple data sets.

10.2.1 Reporting application description

The AspireBI reporting application is a dynamic web project that leverages the power of BIRT and incorporates it to deliver pixel perfect reports. The following diagram shows the architecture of the application:

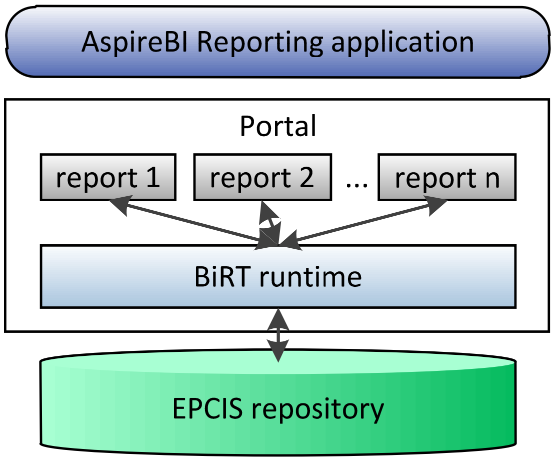


Figure 23 AspireBI logical architecture

The portal (dynamic web project) is used as a container for the reports created by BIRT and uses the BIRT runtime engine to process the reports, retrieve the data through the EPCIS repository and deliver them in various formats as HTML, word document, excel document, power point slides, pdf document or as raw data in csv format. You also have the ability to directly print the reports from the portal.

The EPCIS repository is exposed through a soap web service. First the report generates a soap request that is past down to the repository through the BIRT runtime engine and gets back a soap response, that is an xml data container. The reports get the data from this data container and the presentation is created. In some cases, first an OLAP data source is created from these soap responses and then the reports use the OLAP as a data source for the presentation to be created. An example of such a web service soap envelope for object events is presented below:

```
<soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
     <soap:Body>
          <ns3:Poll

     xmlns:ns2="http://www.unece.org/cefact/namespaces/StandardBusin
essDocumentHeader"
               xmlns:ns3="urn:epcglobal:epcis-query:xsd:1"
xmlns:ns4="urn:epcglobal:epcis:xsd:1"
               xmlns:ns5="urn:epcglobal:epcis-masterdata:xsd:1">
               <queryName>SimpleEventQuery</queryName>
               <params>
                    <param>
                         <name>eventType</name>
                         <value xsi:type="ns3:ArrayOfString"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
                              <string>ObjectEvent</string>
                         </value>
                    </param>
                    <param>
                         <name>EQ_action</name>
                         <value xsi:type="ns3:ArrayOfString"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
                              &?EQ_Action?& </value>
                    </param>
                    <param>
                         <name>GE_eventTime</name>
                         <value xsi:type="ns7:dateTime"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

     xmlns:ns7="http://www.w3.org/2001/XMLSchema">&?GE_EventTime?&</
value>
                    </param>
                    <param>
                         <name>LT_eventTime</name>
                         <value xsi:type="ns7:dateTime"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

     xmlns:ns7="http://www.w3.org/2001/XMLSchema">&?LT_EventTime?&</
value>
                    </param>
                    <param>
                         <name>EQ_disposition</name>
                         <value xsi:type="ns3:ArrayOfString"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
                              &?EQ_Disposition?& </value>
```

```
                        </param>
                        <param>
                              <name>GE_recordTime</name>
                              <value xsi:type="ns7:dateTime"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

      xmlns:ns7="http://www.w3.org/2001/XMLSchema">&?GE_RecordTime?&<
/value>
                        </param>
                        <param>
                              <name>LT_recordTime</name>
                              <value xsi:type="ns7:dateTime"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

      xmlns:ns7="http://www.w3.org/2001/XMLSchema">&?LT_RecordTime?&<
/value>
                        </param>
                        <param>
                              <name>EQ_bizStep</name>
                              <value xsi:type="ns3:ArrayOfString"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
                                    &?EQ_bizStep?& </value>
                        </param>
                        <param>
                              <name>MATCH_epc</name>
                              <value xsi:type="ns3:ArrayOfString"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
                                    <string>&?Match_EPC?&</string>
                              </value>
                        </param>
                        <param>
                              <name>EQ_readPoint</name>
                              <value xsi:type="ns3:ArrayOfString"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
                                    &?EQ_readPoint?& </value>
                        </param>
                        <param>
                              <name>EQ_bizLocation</name>
                              <value xsi:type="ns3:ArrayOfString"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
                                    &?EQ_bizLocation?& </value>
                        </param>
                  </params>
            </ns3:Poll>
      </soap:Body>
</soap:Envelope>
```

Inside the symbols “&? ?&”, a parameter can be defined. Here, all the necessary parameters are implemented with default values that will bring all events. If the client does not want to see prompts for some of them he or she can omit them by deleting everything between the tags <param>"…parameter definition…"</param> of the parameter. By clicking the "Edit Parameter" button, the default parameter values can be defined. For all the parameters except of the date time and quantity parameters, if a value is given that is not enclosed in string tags it will be ignored.

Section 11 Connector

11.1 Overview and purpose

RFID middleware components described in the previous paragraphs provide a foundation for translating raw RFID streams to meaningful business events comprising business context such as where a tag was seen, at what time and in the scope of which process. Enterprises can then leverage these business events through their legacy IT systems (e.g., ERPs, WMS, corporate databases), which are used to support their business processes.

To this end, there is a clear need for interfacing these legacy systems, with the information sharing repositories, established and populated as part of the RFID deployment. Interfacing between IT systems and the information sharing repository (EPCIS), as well as other middleware blocks of the RFID deployment is realized through specialized middleware components that are called "connectors" [2].

The main purpose of connector components is to abstract the interface between the EPC information sharing repository and the enterprise information systems. Hence, connectors offer application programming interfaces (APIs) that enable proprietary enterprise information systems to exchange business information with the ASPIRE RFID middleware system.

A Connector therefore provides:

- **Support for services and events**: Composite applications can call out to existing functionality as a set of services, and to be notified when a particular event type (for example, "purchase order inserted," "employee hired") occurs within an existing application.

- **Service abstraction**: All services have some common properties, including error handling, syntax, and calling mechanisms. They also have common access mechanisms such as JCA (Java Connector Architecture), JDBC, ODBC (Object Database Connectivity), and Web services, ideally spanning different platforms. This makes the services more reusable, while also allowing them to share communications, load balancing, and other non-service-specific capabilities.

- **Functionality abstraction**: Individual services are driven by metadata about the transactions that the business needs to execute.

- **Process management**: Services embed processes, and process management tools call services. Hence, connectors support the grouping of several service invocations to processes.

In this way, a corporation having a legacy IT system can install and communicate with a RFID infrastructure without needing to change the IT infrastructure or significantly alter it. The effort needed to succeed towards the direction of incorporating the RFID infrastructure into the information loop is designed to be minimal and as it is going to be explained later on.

11.2 Current implementation status

The connector component is intended to provide means of transparency between an application and the EPCIS repository. An EPCIS repository is able to collect business level events and provide this information using a push and pull concept. The Connector that we define in this Section is a two-tier component, namely the Connector Engine (CE) and the Connector Client (CC) that operate in the basis of a transaction. These two collaborating tiers enable the user application to receive EPCIS events for specified operations called transactions, using the push or the pull model in a uniform way. By this, we mean that an application can either subscribe to a specific type of operation and when these occur, the application can be notified by the connector, or an application can request information about past observations - defined by time boundaries - of the specific operation.

In the following sections we will describe the tiers that define the Connector interface specification, along with the messages that are used for the internal communication between the tiers and the external communication of the Connector with the EPCIS and with the client application.

11.2.1 The Connector Engine

This tier that is part of the Connector component, has been designed to be "next-to" the EPCIS component. By that, we mean that CE is responsible for the communication of the Connector with the EPCIS repository through the EPCIS Query and Capture Interfaces [10]. The CE tier interface specification also provides a web service interface to receive requests from the CC tier regarding subscription and polling requests for specific transaction type. This interface is extensible and enables the definition of additional operations. The web service specification defines two operations, namely the:

- startObservingTransaction and the
- stopObservingTransaction

Both of them take as an argument a **subscriptionParameters** type. The decision to use this construct in both operations permits us to handle in a uniform way poll or subscription requests.

```
<xs:complexType name="subscriptionParameters">
 <xs:sequence>
  <xs:element name="doPoll" type="xs:boolean"/>
  <xs:element minOccurs="0" name="initialTime" type="xs:dateTime"/>
  <xs:element minOccurs="0" name="queryDayOfMonth"
              type="xs:string"/>
  <xs:element minOccurs="0" name="queryDayOfWeek" type="xs:string"/>
  <xs:element minOccurs="0" name="queryHour" type="xs:string"/>
  <xs:element minOccurs="0" name="queryMin" type="xs:string"/>
  <xs:element minOccurs="0" name="queryMonth" type="xs:string"/>
  <xs:element minOccurs="0" name="querySec" type="xs:string"/>
  <xs:element minOccurs="0" name="replyEndpoint" type="xs:string"/>
  <xs:element name="reportIfEmpty" type="xs:boolean"/>
  <xs:element minOccurs="0" name="subscriptionId" type="xs:string"/>
  <xs:element minOccurs="0" name="transactionId" type="xs:string"/>
  <xs:element minOccurs="0" name="transactionType"
type="xs:string"/>
```

```
 </xs:sequence>
</xs:complexType>
```

A transaction, as described before, can be mapped to a legacy IT system event. By invoking the **startObservingTransaction** operation, the legacy system will be notified for this event. If the observation is based on the push model (subscription) and depending on the defined intervals for checking if the event has occurred, the legacy system will be notified by the CC about this occurrence whenever the time triggers are met. If the observation is based on the pull model (poll), the legacy system will be notified about the past occurrences - limited by the time constraints - immediately.

The invocation of the stopObservingTransaction is designed to be invoked by legacy applications that have previously used the startObservingTransaction operation to subscribe to specific transaction events. This operation handles the cancellation of the subscription on the EPCIS side and the mandatory alteration of specific Master Data in the EPCIS repository through a transaction delete operation. Table 9 contains a description for each of the fields.

11.2.2 The Connector Client

This component has been designed to be located on the side of the legacy IT system and support its interactions with the RFID infrastructure. It is responsible for the following number of operations:

- Provide an interface to the legacy IT systems for receiving query (subscription or polling) requests though a provided application programming interface or through a web service
- Submit the queries to the Connector Engine that it interacts with
- Receive query responses from the CE. The responses may either be a response to a push (subscription) or pull (poll) request, and
- Pass the information to the legacy software

The component provides a web service to receive events from the CE based on subscribed or polled queries. This operation is called asynchronously from the CE component when event information is available and receives an Event object that encapsulates information provided by EPCIS events and is defined in the EPCIS 1.0.1 specification [10]. The following table is the definition of the Event structure in XML format. By encapsulating all the required information within one specific structure instead of four that the EPCIS specification defines, enables legacy IT systems to handle common events captured by the RFID infrastructure with a minimal development, testing and deployment effort.

A legacy IT system wanting to interact with an RFID infrastructure that is connector-enabled would only need to attach to CC by implementing a small number of operations that would handle the Events whenever they occur and by enabling the submission of queries through calls to the CC component.

```
<xs:complexType name="event">
 <xs:sequence>
  <xs:element minOccurs="0" name="action" type="xs:string"/>
  <xs:element minOccurs="0" name="bizLocationId" type="xs:string"/>
  <xs:element minOccurs="0" name="bizStepId" type="xs:string"/>
  <xs:element maxOccurs="unbounded" minOccurs="0"
              name="bizTransactionList" nillable="true"
              type="xs:string"/>
  <xs:element maxOccurs="unbounded" minOccurs="0" name="childEpcs"
              nillable="true" type="xs:string"/>
  <xs:element minOccurs="0" name="dispositionId" type="xs:string"/>
  <xs:element minOccurs="0" name="epcClass" type="xs:string"/>
  <xs:element maxOccurs="unbounded" minOccurs="0" name="epcList"
              nillable="true" type="xs:string"/>
  <xs:element name="eventTime" type="xs:long"/>
  <xs:element minOccurs="0" name="parentId" type="xs:string"/>
  <xs:element name="quantity" type="xs:int"/>
  <xs:element minOccurs="0" name="readPointId" type="xs:string"/>
  <xs:element minOccurs="0" name="subscriptionId" type="xs:string"/>
 </xs:sequence>
</xs:complexType>
```

Table 8 Event Structure Definition in XML format

The client application should implement the ClientEventHandler interface and register itself through the setEventHandler of the ConnectorClientImpl class so that it may receive the events and do whatever it wants.

Field name	**XML Field type**	**Description**	**Use with startObserving Transaction**	**Use with stopObserving Transaction**	**When is required**
doPoll	boolean	This parameter handles that way a request will be processed. If false, then a new subscription will be registered within the EPCIS with information that is provided with other elements. If true then the query will be executed only once and the results will be returned immediately. In any case the **replyEndpoint** will be used to send the result	yes	no	Mandatory for new subscriptions

querySec	String	The category of **queryX** parameters define the time interval that the query will be executed within the EPCIS repository if doPoll is false. At least one of these parameters should be defined in this case. These parameters take a comma separated list of integers that define the query schedule. For example, if querySec has the value *1,31* then the query will be executed on the 1st and 31st second of every minute. Specifies that the query time must have a matching seconds value. The range for this parameter is 0 through 59, inclusive [10]	yes	no	Mandatory for new subscriptions
queryDayOfWeek	String	Specifies that the query time must have a matching day of week value. The range for this parameter is 1 through 7, inclusive, with 1 denoting Monday, 2 denoting Tuesday, and so forth, up to 7 denoting Sunday. This numbering scheme is consistent with ISO-8601 [10]	yes	no	Mandatory for new subscriptions
queryHour	String	Specifies that the query time must have a matching hour value. The range for this parameter is 0 through 23, inclusive, with 0 denoting the hour that begins at midnight, and 23 denoting the hour that ends at midnight [10]	yes	no	Mandatory for new subscriptions
queryMin	String	Specifies that the query time must have a matching minute value. The range for this parameter is 0 through 59, inclusive [10]	yes	no	Mandatory for new subscriptions
queryMonth	String	Specifies that the query time must have a matching minute value. The range for this parameter is 0 through 59, inclusive [10]	yes	no	Mandatory for new subscriptions
queryDayOfMonth	String	Specifies that the query time must have a matching minute value. The range for this parameter is 0 through 59, inclusive [10]	yes	no	Mandatory for new subscriptions
replyEndpoint	String	Specifies that the query time must have a matching minute value. The range for this parameter is 0 through 59, inclusive	yes	no	Mandatory for new subscriptions

reportIfEmpty	boolean	Indicates whether a query result will be send to the Connector client even if there is no matching event	yes	no	Mandatory for new subscriptions
subscriptionId	string	Should be a universally unique identifier to identify a subscription	no	no	Mandatory if doPoll false
transactionId	string	Indicates the events that we are interested in	yes	no	Mandatory for new subscriptions and for deletions
transactionType	string	Indicates the optional event type that we are interested in	yes	no	Mandatory in deletions only if it had been defined in the initial subscription
initialTime	dateTime	This parameter defines the time constraint after which all matching events will be returned. If doPoll is false, and the initialTime refers to the past, the old matching events will be returned within the first response message	yes	no	Mandatory for new subscriptions

Table 9 Subscription parameters fields

The client application should use the RegistrationManager operations to register or execute new queries passing a SubscriptionParameters object and unregister from existing subscriptions.

Section 12 Summary and Conclusions

In this deliverable we have elaborated on the core components of the core ASPIRE middleware infrastructure, providing detailed information for each of these components both to conceptual and practical level. In this deliverable we analyzed the components that constitute a robust system, able to handle realistic deployments and scenarios. The concepts and functionalities developed throughout the project can be considered fully operational and mature.

Starting from the main architecture that was consolidated in the first steps of the project, the constituent components that were developed separately, the overall result can be evaluated as an end-to-end RFID middleware capable of supporting various scenarios, even of high complexity. Moreover, it needs to be noted that compliance to international widely respected standards allows the use of the middleware even based on specific components and not as a whole. It is not necessary for an RFID solution designer, engineer or user to adopt the middleware as a whole. Autonomy in the developed components in combination with the standardized interfaces renders the middleware to a powerful solution for problems in any application layer regarding RFID monitoring.

Additionally, the open source nature of the project offers the possibility to the end user to involve actively in the large and complex development and maintenance process inevitably associated with such large-scale projects. The launch of the AspireRfid open source project and the community formed around it not only eased this procedure but also contributed highly to the shaping of the middleware while in the same time assuring its viability even after the end of the project.

Special reference needs to be made to the licensing issues that occurred during the project's course. These have been effectively resolved after creative reconciliation with the owners of the licensed products and have led to a technology exchange, beneficial for both parties.

For the abovementioned reasons, it can be stated that the development that brought the middleware to its current state has fulfilled its initial goal: ASPIRE is a state-of-the-art, highly competent, flexible and, most importantly, zero cost RFID middleware.

It can be argued, however, that software systems with the complexity and depth of ASPIRE can never be considered complete per se as always new evolutions will present new perspectives in development and evolution. However, the offered functionality has reached a high maturity level, constituting a reliable integrated solution.

The following table summarizes the tools created in the scope of the project in order to provide an overview of the capabilities a user has to communicate/configure the various components.

Tool	**Description**	**Related Middleware component(s)**
EC Spec Editor	Provide an interface to edit ECSpec files (event cycle specifications).	F&C
LR Spec Editor	Provide an interface to edit LRSpec files (logical reader specifications).	F&C

Reporting tools (AspireBI)	Responsible for generating, viewing and exporting reports	EPCIS
Master Data Editor	GUI for editing a Master Data document	EPCIS
Business Event Generator	The role of the BEG is to automate the mapping between reports from F&C and IS events.	F&C and EPCIS
Logical Reader Editor	Allows definition of new readers (HAL, LLRP, RP, or composite)	Reader Management
Programmable Engine	End-to-end middleware configuration	All
BPWME	GUI for the visual business process workflow definition	All
Connector	API to communicate with an EPCIS repository	EPCIS

Table 10 Tools developed and respective architecture components

Section 13 Open issues and Challenges

Throughout this deliverable we have demonstrated and analyzed the work done in the scope of the project regarding the core ASPIRE middleware infrastructure. More specifically, starting from the overall logical architecture, we presented in depth each individual component while also documenting its interfaces with the related components. The result constitutes a robust middleware that can be considered complete and fully functional regarding its capabilities. It must be underlined that the functionality of the final version of the middleware is a result of the merging of the two development branches (AITdev and RFID Suite) that evolved in parallel during most of the project's lifetime. The following list contains the origin of the components available in the final version:

- Provided by AITdev branch
    - Reader Core Proxy
    - ALE server
    - EPCIS
    - BEG
    - Connectors
- Provided by the RFID Suite branch
    - Plug and play sensors
    - EPCIS extensions to store sensor data
    - GWT-based User Console
    - ONS based on Web Services
    - Porting the existing readers (Tagsys, TIRIS, ACS122, Mir:ror) to the reader core proxy
    - Bluetooth bridge as a reader
    - HTTP bridge as a reader
    - NFC MIDLets
    - Management and deployment :
        - LDAP for X509 certificates publication and for architecture description
        - JMX for configuration and monitoring
    - JVisualVM and JConsole plugins for OSGi management (may be contributed to the Apache Felix community).

Moreover, because of the licensing issues that were raised regarding reuse of certain Fosstrak components, the consortium reevaluated the option of using licensed software and investigated the need and possibility of using exclusively consortium developed components. Finally, after negotiation and collaboration with the Fosstrak developing team, numerous enhancements and bugfixes were realized and committed back to Fosstrak. As a result, full compatibility was maintained between the two versions, always compliant to EPCglobal specifications.

However, as analyzed next per component, much space remains for further improvement.

13.1 Infrastructure components

The ONS approach that uses Web services is an interesting add-on to the standard DNS based EPC Global ONS, providing a high level interface for application developers that need to interact with the AspireRfid middleware. However, data governance is an important aspect on this part of the system.

Evaluation of impact and priority concerning the parts of the ASPIRE middleware could be modularized so they can take advantage of both the OSGi technology and the Apache Felix iPOJO component model.

Exposing other system functionalities as JMX MBean interfaces would allow the control of ASPIRE middleware components centralized in the ASPIRE IDE. Functionalities for resetting and configuring other applications, for example, would minimize the user effort from switching between different applications.

13.2 Supported readers

As far as it concerns the readers supported by the middleware, Section 5 analyzed in detail the variety of the supported readers. However, given the width of choice regarding RFID readers, support can be considered as a constantly ongoing procedure. In addition, since standards also evolve, future steps can include porting the existing drivers in order to be compliant with the latest version of EPCglobal Reader Protocol standard (currently version 1.1) [21].

Regarding NFC readers, future steps could include the integration of the NFC library and the Bluetooth bridge as readers compliant with the EPCglobal Reader Protocol standard [21]. The library will be completed by utilities components querying the future version of the ASPIRE ONS. Moreover, NFC phones could query the ONS provided by ASPIRE using Web services (RESTful or SOAP based).

13.3 Filtering and Collection

The Fosstrak's implementation of the ALE specification is not complete and it misses a very significant part of the specification, called Writing API. This application programming interface provides a standardized way of writing data onto the RFID chip memory and can be extremely helpful in specific use cases.

13.3.1 Distributed F&C

For now, the distributed ALE deals only with non-composite Logical Readers and it is easy to add support for composite logical readers composed only by local physical readers. Providing mechanisms allowing the declaration and the use of distributed logical readers will be interesting in order to propose real insulation for business applications. Along with a mapping between virtual coordinates of nodes in the network and geographical real positions, it provides a way to associate all physical distributed readers of a geographical region into a single logical reader.

13.3.2 Embedded F&C

The CODEC subset and, consequently, the TagGenerator application, only support sgtin-96 and sgln-96 for now. Additional support for other tag encoding would be useful.

As for the ALE implementation, full support of the Reading API may be useful for other scenarios. Moreover, XML rendering on PDA would be interesting for maintaining EPC global specification files. Actually, the API was written manually. Generating J2ME compliant class files from specification XSD files would ease the development and maintenance process. This may be achieved using a tool like JAXB (which has no support for J2ME yet).

Finally, the Reader Protocol implementation is minimal for inventory. Adding separate MTB layers may be useful when porting to an RP-compliant reader.

13.3.3 Sensor data inclusion

As far as it concerns the inclusion of sensor data (see Section 6.6), future steps are to express new functions to produce set of reports. Actually, there are possibilities to produce the reports:

- CURRENT: the report is produced with current read tags.
- ADDITIONS: the report is produced with read tags in an event cycle relative to the previous one.
- DELETIONS: the report is produced with non-present tags relative to the previous one.

The extension consist in add the possibility to express more complex policies in order to produce the set of tags to be included in the reports. These policies may be expressed using a Script language (Javascript) or a traditional programming language (Java).

13.4 Business Event Generator

Additionally to the already implemented fully functional features, the Business Event Generator component can still be augmented with some new ones and even be subjected to changes at its current design, as long as the functionality is preserved, as dictated by the respective standards.

Some of the possible changes/additions is the creation of an interface for connecting/supporting:

- Actuators that will be able to connect to the AspireRfid architecture through the BEG component and e.g. support a two way interaction with a “warehouse”.
- Interface for giving feedback to various devices (e.g. Account machines, delivery Information at the gates/handheld devices)

Another probable change is that the BEG component will get implemented as an OSGi bundle that can be deployed on top of a JOnAS Application Server which is both a JavaEE container and a OSGi container.

Finally, a necessary addition to the BEGs features is to provide appropriate authentication and access mechanisms for permitting or denying management and configuration access for its various interfaces.

13.5 EPCIS

Besides the already implemented functionality in the Information Sharing repository there is still margin for improvement. Some of the additional functionality that can be implemented is the following:

- Addition of new Field extensions for the existing Event Types extensions in the Data Definition Layer (Useful to store e.g. GPS coordinates, temperature, hygrometry, shock events, survey answers, security check, etc.).
- Investigation of implementing an EPCIS repository as an OSGi bundle that can be deployed in JOnAS Application Server.
- Implementation of EPCIS specifications, and probably extension of the authentication and access mechanisms.

13.6 Connector client

The connector concept has yet to be validated in real world situations. Such situations would involve widely adopted legacy IT systems being able to communicate with an RFID infrastructure through the connector component. The communication process should involve:

- The IT system to be able to register for specific business events in the EPCIS repository
- The EPCIS component to be able to send business events to registered clients

In order to be able to test the validity of the connector concept and design, connectors can be developed for major legacy IT systems the respective operational behavior in real cases can be evaluated. Then, having the evaluation results in hand we will be able to make any required amendments and/or define extensions to the connector interface defined, to accommodate the required functionality. This process will iterate until we have concrete evidence through the validation process that every major requirement based on the legacy IT systems needs has been accommodated by the connector design.

List of Figures

List of Tables

List of Acronyms

ALE	Application Level Event
APDL	AspireRfid Process Description Language
API	Application Program Interface
ASPIRE	Advanced Sensors and lightweight Programmable middleware for Innovative Rfid Enterprise applications
BEG	Business Event Generator
ECA	Event-Condition-Action
EDI	Electronic Data Interchange
EPC	Electronic Product Code
EPCIS	EPC Information Service
ERP	Enterprise Resource Planning
F&C	Filtering and Collection
GUI	Graphical User Interface
HAL	Hardware Abstraction Layer
HTTP	Hyper Text Transfer Protocol
IDE	Integrated Development Environment
JMS	Java Messaging Service
LLRP	Low Level Reader Protocol
MTB	Message Transport Binding
ONS	Object Name Service
OSGi	Open Services Gateway Initiative
RP	Reader Protocol
SME	Small Medium Enterprise
TDS	Tag Data Standard
TDT	Tag Data Translation
URL	Uniform Resource Locator
WMS	Warehouse Management System
XML	eXtensible Markup Language

References

[1] L. Schmidt, N. Mitton and D. Simplot-Ryl. *Towards Unified Tag Data Translation for the Internet of Things*. In Wireless Communication Society, Vehicular Technology, Information Theoryand Aerospace & Electronics Systems Technology (VITAE'09), Aalborg, Denmark, 2009.
[2] A. Gallais and J. Carle. *Performance Evaluation and Enhancement of Surface Coverage Relay Protocol*. In Proc. IFIP Networking'08 - Singapore, May 2008.
[3] A. Gallais, F. Ingelrest, J. Carle and D. Simplot-Ryl. *Preserving Area Coverage in Sensor Networks with a Realistic Physical Layer*. In Proc. IEEE INFOCOM'07 - Anchorage, Alaska, May 2007.
[4] A. Gallais, J. Carle, D. Simplot-Ryl and I. Stojmenovic *Ensuring K-Coverage in Wireless Sensor Networks with Realistic Physical Layers*. In Proc. IEEE Sensors'06 - Daegu, Korea, October 2006.
[5] Mihaela Cardei, My T. Thai, Yingshu Li, Weili Wu *Energy-Efficient Target Coverage in Wireless Sensor Networks* Proceedings of 24th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), Vol. 3, Miami, FL, United States, March, 1976-1984
[6] EPCglobal Low Level Reader Protocol standard, available online at http://www.epcglobalinc.org/standards/llrp
[7] LLRP-Toolkit - http://www.llrp.org/
[8] EPCglobal Application Level Events standard, available online at http://www.epcglobalinc.org/standards/ale
[9] Fosstrak project - http://www.fosstrak.org/
[10] EPCglobal Electronic product Code Information Services (EPCIS) standard, available online at http://www.epcglobalinc.org/standards/epcis
[11] EPCglobal Object Name Service (ONS) standard, available online at http://www.epcglobalinc.org/standards/ons
[12] NFC Forum Specification http://www.nfc-forum.org/specs/
[13] JSR 257: Contactless Communication API http://jcp.org/en/jsr/detail?id=257
[14] OSGi Specifications http://www.osgi.org/Specifications/HomePage
[15] Apache Felix iPOJO http://felix.apache.org/site/apache-felix-ipojo.html
[16] ASPIRE Internal Report D2.4, ASPIRE Middleware and Programmability Specifications
[17] Fagui Liu, Yuzhu Jie, and Wei Hu. Distributed ale in rfid middleware. In Wireless Communications, Networking and Mobile Computing, 2008. WiCOM '08. 4th International Conference on, pages 1–5, Oct. 2008
[18] Jae Geol Park, Heung Seok Chae, and Eul Seok So. A dynamic load balancing approach based on the standard rfid middleware architecture. In ICEBE '07: Proceedings of the IEEE International Conference on e-Business Engineering, pages 337–340, Washington, DC, USA, 2007. IEEE Computer Society
[19] EPCglobal, The EPCglobal Architecture Framework Version 1.3, available online at http://www.epcglobalinc.org/standards/architecture/
[20] Spring Framework http://www.springsource.org/
[21] EPCglobal: Reader Protocol standard, available online at http://www.epcglobalinc.org/standards/rp

Appendix A – Filtering and Collection component (based on Fosstrak implementation) user guide

1. Requirements

Hardware (minimum)

- P IV 1.2GHz or equivalent
- 512 MB Ram
- 50 MB free HD space

Software

- Java 1.6
- Tomcat 5.5 (or higher) or another server for web-services. (This guide assumes that you use an Apache Tomcat server.)

2. Deployment

Copy the aspireRfidALE.war file which can be found at the ApireRFID Forge into the webapps folder of your server and start the server. The war file will be deployed into a new folder. Under windows you will usually find the webapps folder inside the tomcat installation directory (for instance c:\Program Files\Apache Tomcat\webapps). Under linux/unix this will depend on your distribution. Some possible locations:

- /var/lib/tomcat/webapps
- /usr/local/lib/tomcat/webapps

The ALE server is now ready to be configured at your needs.

3. Configuration

This section gives a short overview to the configuration files available. These files allow you to adapt the aspireALE to your needs. You can find these configuration files inside the folder TOMCAT_DIRECTORY/webapps/aspireALEVERSION/WEB-INF/classes.

example: /var/lib/tomcat/webapps/aspireALE0.3.1m/WEB-INF/classes

InputGenerators.properties: This properties-file is the main configuration file for the ASPIRE ALE. You can find it in the Folder WEB-INF/classes. It allows only one parameter to be changed, namely the xml-file that provides the logical reader API with the initial readers available at startup.

LogicalReaders.xml: This file specifies the readers that are loaded during startup of the ALE.

After a restart of the webserver the ASPIRE Filtering and Collection is available and ready to accept client connections.

4. Logical Reader Configurations

This guide introduces Logical Readers and how they can be declared to be used in the Filtering and Collection server.

There are two different types of Logical Reader Definitions that should not be confused:

Dynamic Logical Reader Definitions: Dynamic Logical Reader Definitions are read by the ALE Configurator. If you want to specify a logical reader at runtime through the Logical Reader API you need to use a Dynamic Logical Reader.

Static Logical Reader Definitions: Static Logical Reader Definitions are read/written by the Logical Reader Manager upon Filtering and Collection server deployment. They contain additional information for the Logical Reader Manager.

4.1 LogicalReaders

LogicalReaders act always either as a connector between software and hardware or as a connector between software and software. Therefore you need some parameters that configure your LogicalReader at your needs. In the following we will give a short introduction how you can setup the basic structure for a LogicalReader.

When you want to define your own LogicalReader through an xml file you need to obey some restrictions. Some of them are discussed here.

- The xml must have a valid encoding and version number

```
<?xml version="1.0" encoding="UTF-8"?>
```

Dynamic Definition

- The xml must contain exactly one LRSpec definition.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns3:LRSpec xmlns:ns2="urn:epcglobal:ale:wsdl:1"
        xmlns:ns3="urn:epcglobal:ale:xsd:1">
</ns3:LRSpec>
```

- You must define whether the reader is composite or not.
- The reader must contain at least the LRProperty of the ReaderType.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns3:LRSpec xmlns:ns2="urn:epcglobal:ale:wsdl:1"
        xmlns:ns3="urn:epcglobal:ale:xsd:1">
    <isComposite>false</isComposite>
    <readers/>
    <properties>
```

```
        <property>
            <name>ReaderType</name>
            <value>
org.ow2.aspirerfid.ale.server.readers.hal.HALAdaptor</value>
        </property>
</ns3:LRSpec>
```

- If your reader is a composite reader, you must provide the list of the "subreaders".

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns3:LRSpec xmlns:ns2="urn:epcglobal:ale:wsdl:1"
        xmlns:ns3="urn:epcglobal:ale:xsd:1">
    <isComposite>true</isComposite>
    <readers>
        <reader>LogicalReader1</reader>
    </readers>
    <properties>
        <property>
            <name>ReaderType</name>
            <value>
org.ow2.aspirerfid.ale.server.readers.CompositeReader</value>
        </property>
    </properties>
</ns3:LRSpec>
```

Static Definition

- The xml must contain exactly one LogicalReaders tag.

```
<?xml version="1.0" encoding="UTF-8"?>
<LogicalReaders xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
      xsi:noNamespaceSchemaLocation="/resources/LogicalReaders.xsd">
</LogicalReaders>
```

- Whenever you define a LogicalReader you must specify an LRSpec and within that LRSpec you must specify if this reader is composite or not.

```
<?xml version="1.0" encoding="UTF-8"?>
<LogicalReaders xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
      xsi:noNamespaceSchemaLocation="/resources/LogicalReaders.xsd">
  <LogicalReader name="LogicalReader1">
    <LRSpec isComposite="false"
    readerType="org.ow2.aspirerfid.ale.server.readers.rp.RPAdaptor">
    </LRSpec>
  </LogicalReader>
```

```
</LogicalReaders>
```

- Make sure, that you use the name of a LogicalReader only once. The logical reader API does not allow duplicates of LogicalReaders.

Appendix B – EPCIS (based on Fosstrak Implementation) users guide

1. Requirements

Hardware (minimum)

- P IV 1.2GHz or equivalent
- 512 MB Ram
- 50 MB free HD space

Software

- Java 1.6
- Tomcat 5.5 (or higher) or another server for web-services. (This guide assumes that you use an Apache Tomcat server.)
- MySQL 5.0 (or higher).

2. Deployment

This section includes a step-by-step tutorial describing how to set up your own EPCIS repository.

In order to set up your own EPCIS repository, you have to follow the following steps: Firstm make sure you have an Apache Tomcat servlet container (version 5.5 or higher) running. It will be used to deploy and run the EPCIS repository web application. Download the latest aspireRfidEpcisRepository distribution found at the ASPIRE's Forge and place the WAR file contained in the archive in your Tomcat's webapps directory. After restarting Tomcat, the WAR file will be exploded (i.e. its contents will be uncompressed into a directory under webapps). Install a MySQL server (version 5.0 or higher). It will be used by the EPCIS repository to store event data. Make sure that web applications deployed to Tomcat can access your MySQL server by installing the MySQL Connector/J driver. This is usually done by copying the mysql-connector-java--bin.jar into Tomcat's lib (version 6) or common/lib (version 5.5) directory. Set up a MySQL database for the EPCIS repository to use. Log into the MySQL Command Line Client as root and perform the following steps:

Create the database (in this example, we will use epcis as the database name).

```
mysql> CREATE DATABASE epcis;
```

Create a user that is allowed access to the newly created database (in this example, we will use the user name epcis and password epcis).

```
mysql> GRANT SELECT, INSERT, UPDATE, DELETE, DROP, CREATE, ALTER ON
epcis.* TO epcis IDENTIFIED BY 'epcis';
```

Create the database schema by running the "epcis_schema.sql" script contained in the archive you downloaded (located at: aspireRfidEpcisRepository/WEB-INF/classes/sql folder). (Make sure you are connected to the newly created database before running the script.)

```
mysql> USE epcis;
mysql> SOURCE <path-to-unpacked-download>/epcis_schema.sql
```

Optionally populate the repository with some sample data.

```
mysql> SOURCE <path-to-unpacked-download>/epcis_demo_data.sql
```

Configure the repository to connect to the newly created database. In a default installation of Tomcat, the database connection settings can be found in $TOMCAT_HOME/conf/Catalina/localhost/aspireRfidEpcisRepository.xml. The relevant attributes that must be adjusted are username, password, and url.

```
<Resource
    name="jdbc/EPCISDB"
    type="javax.sql.DataSource"
    auth="Container"
    username="epcis"
    password="epcis"
    driverClassName="org.gjt.mm.mysql.Driver"
    url="jdbc:mysql://localhost:3306/epcis?autoReconnect=true">
</Resource>
```

If you used the default user name, password and database name from the examples above, then you don't need to reconfigure anything here. If, however, you used different values, you need to stop Tomcat, change the values and start Tomcat again.

Check if the application is running. In a default installation of Tomcat, the capture and query interfaces will now be available at http://localhost:8080/aspireRfidEpcisRepository/capture and http://localhost:8080/aspireRfidEpcisRepository/query respectively.

When you open the capture interface's URL in your web browser, you should see a short information page similar to this:

This service captures EPCIS events sent to it using HTTP POST requests. The payload of the HTTP POST request is expected to be an XML document conforming to the EPCISDocument schema.

For further information refer to the xml schema files or check the Example in 'EPC Information Services (EPCIS) Version 1.0 Specification', Section 9.6 [10].

To also check if the query interface is set up correctly, point your browser to its URL and append the string ?wsdl to it. The WSDL file of the query service should now be displayed in your browser.

Proceed to the next sections to test your repository installation using one of our client applications.

Check the application's log file in case of problems. The application's log is kept in TOMCAT_HOME/logs/aspireRfidEpcisRepository.log. In case of problems with your own EPCIS repository instance, this is the first place to look for information about errors or specific exceptions thrown by the application.

3. Runtime Configuration of your EPCIS Repository

In this section, we describe the properties you can use to configure AspireRfid EPCIS repository implementation.

Basically there are three coniguration files relevant to the user of the application: application.properties, context.xml, and log4j.properties

1. application.properties The application.properties file is located in the application's class path at TOMCAT_HOME/webapps/aspireRfidEpcisRepository/WEB-INF/classes. It contains the basic configuration directives that control the repository's behaviour when processing queries and events. This file looks as follows:

```
# application.properties - various properties (loaded at runtime)
which are used to configure the behaviour of the epcis-repository
application

# the version of this service, as exposed by getVendorVersion (must
be valid URI)

service.version=http://wiki.aspire.objectweb.org/xwiki/bin/view/Main
.Documentation/EpcisRepository

# maximum number of result rows allowed for a single query before a
QueryTooLarge exception is raised

maxQueryResultRows=1000

# maximum time in milliseconds to wait for a query to finish before
a QueryTooComplex exception is raised

maxQueryExecutionTime=20000

# whether to allow inserting new vocabularies when they are missing
in the db

insertMissingVoc=true
```

```
# the schedule used to check for trigger conditions - the values
provided here are parsed into a query schedule which is used to
periodically check whether incoming events contain a specific
trigger URI or not

trigger.condition.check.sec=0,20,40
trigger.condition.check.min=

# whether to allow resetting the database via a HTTP POST 'dbReset'
parameter

dbResetAllowed=false
dbResetScript=/sql/reset_epcis_with_demo_data.sql

# the location of the EPCglobal EPCIS schema

epcisSchemaFile=/wsdl/EPCglobal-epcis-1_0.xsd

# the location of the EPCglobal EPCIS MasterData schema(nkef)

epcisMasterDataSchemaFile=/wsdl/EPCglobal-epcis-masterdata-1_0.xsd

# whether to trust a certificate whose certificate chain cannot be
validated when delivering results via Query Callback Interface

trustAllCertificates=false

# the name of the JNDI datasource holding the connection to the
database

jndi.datasource.name=java:comp/env/jdbc/EPCISDB
```

We would like to outline one specific feature: The AspireRfid EPCIS implementation includes the option to specify an SQL script (see dbResetScript property) and trigger the execution of this script remotely. This behaviour is not part of the EPCIS specification, but can be used to remotely initialize a repository to a predefined state. The script is triggered by sending an HTTP POST request to the capture interface with the HTTP parameter dbReset set to true. Please note that this feature is not protected by any security mechanisms. It is intended for internal use only and therefore disabled by default (future versions may provide more sophisticated remote management capabilities).

2. context.xml The context.xml file includes the configuration parameters for the database connection and looks as follows:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Context reloadable="true">
  <Resource name="jdbc/EPCISDB"
            type="javax.sql.DataSource"
            auth="Container"
            username="epcis"
            password="epcis"
            driverClassName="org.gjt.mm.mysql.Driver"
            defaultAutoCommit="false"
            url="jdbc:mysql://localhost:3306/epcis?autoReconnect=true">
  </Resource>
</Context>
```

This file is located at TOMCAT_HOME/webapps/aspireRfidEpcisRepository/META-INF/. However, as indicated before, Tomcat reads these configuration settings from the conf/Catalina/localhost/aspireRfidEpcisRepository.xml file once your application has been deployed.

3. log4j.properties This file is also located in the application's class path at TOMCAT_HOME/webapps/aspireRfidEpcisRepository/WEB-INF/classes. The properties defined here affect the logging behaviour of the application. The log file is written to TOMCAT_HOME/logs/aspireRfidEpcisRepository.log. By default, it only includes log statements of level INFO and higher.

Appendix C – Connector component Users Guide and Developer Guide

1. Deployment

Both connector components are designed to work in a Java application server (e.g. tomcat) so they are provided as a web archive (i.e. war). The connector client can also be embedded within a client application if required. This means that it can function either within a web container or in standalone mode, functionality that can be selected through configuration files. Instructions on how to deploy the provided war files are dependent on the application server that you may choose to use. For the latest version of tomcat, all you have to do is place a copy of the war files within the webapps directory located at the tomcat installation folder.

2. Configuration

Configuration files exist within the war files under the folder props with the name application.properties. You can either edit them before deployment or after deployment based on your needs and on the application server that you are using. Following we will describe the configuration options for each connector component.

3. Connector server

This is the configuration file of the connector server. We will explain each one of the configuration options.

```
callbackDestinationUrl=http://localhost:8899
epcisQueryIfceUrl=http://localhost:8080/epcis/query
epcisCaptureIfceUrl=http://localhost:8080/epcis/capture
queryName=SimpleEventQuery
timeDifferenceFromUTC=+02:00
```

callbackDestinationUrl: This generally does not need to be changed. It defines a TCP server where query subscriptions to the EPCIS repository will return their results to. If this is defined incorrectly the connector will not be able to receive any query results from the EPCIS repository.

epcisQueryIfceUrl: This property holds the location of the EPCIS query interface that we are interested in getting information from, as defined by the EPCglobal EPCIS standard

epcisCaptureIfceUrl: This is the URL of location of the EPCIS capture interface.

queryName: The only available query name is the SimpleEventQuery. Unless another query is implemented at the EPCIS query interface, this should not be changed.

timeDifferenceFromUTC: This is the time difference of the local time from the UTC or GMT. It is required for the generation of specific EPCIS events. The format is ±HH:MM.

4. Connector client

```
connectorServerUrl = http://localhost:8080/Connector-1.0/connector
```

connectorServerUrl: The endpoint of the connector server. The first part, i.e. http://localhost:8080/Connector-1.0 is the location where the connector server has been deployed. The second part, i.e. connector is the web service that the client uses to communicate with the server and subscribe for new queries.

isConnectorClientStandaloneModeOn: Possible values: true or false. This defines whether the connector client is deployed as a standalone web application or is embedded within a client application. If this property is set to false and the war file is deployed to an application server, an embedded servlet will be used to deploy the web services of the connector client. On the other hand, if this property is set to true, an embedded trivial application server will be opened by the connector client, on the port defined by the standaloneConnectorClientPort property, and the web service will be deployed on this server.