



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 (Interim Version)

Due date of deliverable: M18
 Actual Submission date: 22.06.09

Deliverable ID:	WP3/D3.4a
Deliverable Title:	Core ASPIRE Middleware Infrastructure (Interim Version)
Responsible partner:	AIT
Contributors:	John Soldatos, Nikos Kefalakis, Nektarios Leontiadis – AIT Nathalie Mitton, Loïc Schmidt - INRIA Mathieu David - AAU Didier Donsez, Kiev Gama – UJF
Estimated Indicative Person Months:	12

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

Revision: 1.4
 Dissemination Level: PU

PROPRIETARY RIGHTS STATEMENT

This document contains information, which is proprietary to the ASPIRE Consortium. Neither this document nor the information contained herein shall be used, duplicated or communicated by any means to any third party, in whole or in parts, except with prior written consent of the ASPIRE consortium.

Document Information

Document Name: Core ASPIRE Middleware Infrastructure (Interim Version)
Document ID: WP3/D3.4a
Revision: 1.4
Revision Date: 22 June 2009
Author: AIT
Security: PU

Approvals

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

Reviewers

Name	Organization	Date	Comments	Visa
<i>Mathieu David</i>	CTID-AAU	18 Jun 09		
<i>Didier Donsez</i>	UJF	18 Jun 09		
<i>Ramiro Robles</i>	IT	19 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 6	Loïc Schmidt
0.3	29 may 09	Section 8.2	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 9	Nikos Kefalakis
0.7	15 Jun 09	Section 11	Mathieu David
0.8	16 Jun 09	Paragraphs 10.2, 10.3	Nikos Kefalakis
0.9	16 Jun 09	Section 5, 7.2, 7.3	Kiev Gama, Didier Donsez
1.0	17 Jun 09	Section 1	Nektarios Leontiadis

Contract: 215417
Deliverable report – WP3 / D3.4a

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	19 Jun 09	Extended final version	Ramiro Robles

Content

Section 1	Executive Summary	7
Section 2	List of Acronyms	9
Section 3	Introduction	11
Section 4	Overview of core ASPIRE middleware infrastructure	13
Section 5	Infrastructure components	15
5.1	OSGi (formerly known as Open Service Gateway Initiative)	15
5.1.1	Overview and purpose	15
5.1.2	Interfaces to other components	15
5.1.3	Current implementation status	15
5.1.4	Future steps	15
5.2	Application server	15
5.2.1	Overview and purpose	15
5.2.2	Interfaces to other components	15
5.2.3	Current implementation status	16
5.2.4	Future steps	16
5.3	Java Management Extensions (JMX)	16
5.3.1	Overview and purpose	16
5.3.2	Interfaces to other components	16
5.3.3	Current implementation status	16
5.3.4	Future steps	16
Section 6	Tag Data Translation (TDT)	17
6.1	Overview and purpose	17
6.2	Interfaces to other components	18
6.3	Current implementation status	18
6.4	Summary and future work	18
Section 7	Reader interfaces	20
7.1	Low Level Reader Protocol (LLRP) interface	20
7.1.1	Overview and purpose	20
7.1.2	Interfaces to other components	20
7.1.3	Current implementation status	20
7.2	Supported Readers	20
7.2.1	Overview and purpose	20
7.2.2	Interfaces to other components	20
7.2.3	Current implementation status	20
7.2.4	Future steps	21
7.3	Near Field Communications (NFC)	21
7.3.1	Overview and purpose	21
7.3.2	Interfaces to other components	21
7.3.3	Current implementation status	21

7.3.4	Future steps	21
Section 8	<i>Filtering and Collection (F&C)</i>	22
8.1	Infrastructure	22
8.1.1	Overview and purpose	22
8.1.2	Interfaces to other components	22
8.1.3	Current implementation status	22
8.1.4	Future steps	22
8.2	Anti-collision protocol	23
8.2.1	Overview and purpose	23
8.2.2	Current investigation status	23
8.2.3	Future steps	24
Section 9	<i>Business Event Generator (BEG)</i>	25
9.1	Overview and purpose	25
9.2	Interfaces to other components	26
9.3	Current implementation status	26
9.4	Future steps	27
Section 10	<i>Electronic Product Code Information Services (EPCIS)</i>	28
10.1	Overview and purpose	28
10.2	Current implementation status and interfaces to other components	28
10.2.1	Master Data Capture Interface	29
10.3	Future steps	30
Section 11	<i>Object Name Service (ONS)</i>	31
11.1	Overview and purpose	31
11.2	Interfaces to other components	32
11.3	Current implementation status	32
Section 12	<i>Context analysis</i>	33
12.1	Overview and purpose	33
12.2	Interfaces to other components	33
12.3	Current implementation status	34
12.4	Future steps	34
Section 13	<i>Connector</i>	35
13.1	Overview and purpose	35
13.2	Current implementation status	36
13.2.1	The Connector Engine	36
13.2.2	The Connector Client	37
13.3	Future steps	42
Section 14	<i>Evolution of Aspire middleware infrastructure (D3.4b)</i>	43
Section 15	<i>Conclusions</i>	44
Section 16	<i>List of Figures</i>	45

Section 17	List of Tables.....	46
Section 18	References and bibliography.....	47
Section 19	Appendix A – Filtering and Collection component (based on Fosstrak implementation) user guide	48
19.1	Requirements	48
19.2	Deployment	48
19.3	Configuration.....	48
19.4	Logical Reader Configurations	48
19.4.1	LogicalReaders	49
Section 20	Appendix B – EPCIS (based on Fosstrak Implementation) users guide ..	51
20.1	Requirements	51
20.2	Deployment	51
20.3	Runtime Configuration of your EPCIS Repository	53
Section 21	Appendix C – Connector component Users Guide and Developer Guide	55
21.1	Deployment	55
21.2	Configuration.....	55
21.2.1	Connector server	55
21.2.2	Connector client.....	55

Section 1 Executive Summary

Recent advances in low cost microelectronics and radiofrequency transceivers have paved the way for new applications of radio frequency identification (RFID) systems in the fields of supply chain management and inventory control. In contrast to conventional RFID applications such as toll payment and access control, in which the design of the middleware platform represented a simple and unchallenging task, in the new applications complex interactions between different layers, network components, business contexts, and security and privacy issues must be considered. As a consequence of this and despite the cost of tags and readers are being constantly reduced, middleware platforms entry costs for small and medium enterprises (SMEs) are still high thus limiting the expected widespread adoption of RFID.

ASPIRE is developing an innovative, royalty free and privacy friendly middleware platform that will allow SMEs reducing the total cost associated with the deployment of RFID systems. This middleware platform is a primary target of the open source “AspireRfid” project, which has been recently established in the scope of the OW2 community. The open nature of the “AspireRfid” project asks for versatility in terms of the hardware and software that will support the RFID solutions that will be built based on the ASPIRE middleware platform.

In this deliverable, we will describe developments related to the core Aspire middleware infrastructure until the month 18 of the project. These components consist of both licensed and consortium developed source code and are based on the Aspire architecture specification from deliverables D2.3 and D2.4, which in turn were based on the review of state-of-the art RFID middleware platforms presented in D2.1 and the end-user requirements collected and analyzed in D2.2.

Being fully in line with the Aspire middleware architecture, the design in 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 (Integrated Development Environment), which is being developed in the scope of Work Package 4 (WP4). In principle, ASPIRE middleware should be able to operate with any reader platform regardless of vendors, frequency and supported functionality. Likewise, the ASPIRE middleware should support different tag formats and legacy IT (Information technology) information system languages. This freedom of choice is perfectly in line with both the “open” nature of the middleware and the requirements of the SMEs. 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 does 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 RFID filtering and data processing components, namely the ALE (application level event) and BEG (business event generator) servers, and context analysis components. Finally, the data provisioning components should cater for the diversity of end user applications that require the clean and intelligently processed information. The end user application would require this infrastructure to be able to “talk” in their own and understandable language.

The following is a list of the core Aspire middleware components which are either existing or planned. The developed components are publicly available through the project’s community forge at OW2 (<http://forge.ow2.org/projects/aspire/>) and are divided into two development

branches and are currently under the process of unification. The basic infrastructure of the Aspire technical developments along with the open nature of the components' distribution network, enable the creation of an open community which lead to wider adoption by users and developers.

- Tag data translation
- Reader virtualization
- Filtering and collection
- Business events generator
- EPC information system
- Object naming service
- Context analysis
- Connectors

We will finally need to emphasize on the interim nature of this document, which intends to report the development progress by the time of delivery of this document, and thus, some components might slightly or significantly change in their implementation and configuration.

Section 2 List of Acronyms

AAU	Aalborg University
AIT	Athens Information Technology
ALE	Application Level Event
API	Application Program Interface
ASPIRE	Advanced Sensors and lightweight Programmable middleware for Innovative Rfid Enterprise applications
BEG	Business Event Generator
BTB	Bluetooth Bridge
CC	Connector Client
CE	Connector Engine
CRM	Customer Relationship Manager
DNS	Directory Name Service
ECA	Event condition actions
EDI	Electronic Data Interchange
EPC	Electronic Product Code
EPCIS	Electronic Product Code Information Services
ERP	Enterprise Resource Planning
F&C	Filtering and Collection
GIAI	Global Individual Asset Identifier
GLN	Global Location Number
GPS	Global Positioning System
GRAI	Global Returnable Asset Identifier
GS1	Global Standard 1 (Standardisation group)
GTIN	Global Trade Identification Number
HAL	Hardware Abstraction Layer
HTTP	HiperText Transfer Protocol
IDE	Integrated Development Environment
IP	Internet Protocol
IS	Information System or Information Service
ISO	International Standard Organization
IT	Information Technology
IT	Instituto de Telecomunicações
J2ME	Java 2 Micro Edition
JAS	Java Application Server
JaxWS	Java web server
JCA	Java Connector Architecture
JCP	Java Community Process
JMX	Java Management Extensions
JMS	Java Messaging Service
JVM	Java Virtual Machine
LGPL	Lesser General Public License
LLRP	Low Level Reader Protocol
MSS	Monitoring Service Set
NFC	Near Field communications
ODBC	Object Database Connectivity
ONS	Object Name Service
OSGI	(formerly known as Open Service Gateway Initiative)

OSI	Open System Interconnection
OSS	Open Source Software
OW2	Open source community which is the merge of the ObjectWeb Consortium and Orientware)
RDBMS	Relational database management system
RFC	Request For Comments
RFID	Radio Frequency Identification
RP	Reader Protocol
SME	Small and Medium Enterprise
SNMP	Simple Network Management Protocol
SMTP	Simple Mail Transfer Protocol
SOAP	Simple Object Access Protocol
SSCC	Serial Shipping Container Code
SVN	Subversion
TDS	Tag Data Standard
TDT	Tag Data Translation
TCO	Total Cost of Ownership
TCP	Transfer Control Protocol
UJF	University Joseph Fourier
UML	Universal Mark-up Language
URI	Uniform Resource Identifier
URN	Uniform Resource Name
WMS	Warehouse Management System
WP	Work Package
XML	Extensible Markup Language

Section 3 Introduction

Radio frequency identification (RFID) systems appeared in their first commercial format at the end of the 1960s after an evolution process that started during the Second World War by using the concepts of radar technology and the theory of reflected power. However, widespread adoption of RFID would have to wait until the end of the 1970s and beginnings of the 1980s in systems dedicated to access control and toll payments, which mainly used transceivers in low and high frequency bands. These conventional systems simply consisted of readers or interrogators, which request information from tags; tags or transponders, which respond to reader's requests; and simplified middleware and end processing servers which present the meaningful information to the end user.

The last three decades, however, have seen a rapid development of low cost microelectronics and radio frequency transceivers that have considerably reduced size and costs of high, ultra-high and microwave frequency transceivers for RFID, and which, due to its nature, allow longer reading ranges and faster reading rates than previous systems. These facts further allow new RFID applications with higher mobility and with a larger number of tagged items, which are perfect for supply chain management and inventory control environments. Unlike conventional scenarios, these new applications require a more robust and complex middleware platform in order to cover issues at different layers of the communication architecture, from different business contexts and that must consider new issues on security and privacy domains. This complexity has left several open research issues in RFID middleware design that still pose a high entry cost for RFID technology adopters, mainly SMEs.

The main goal of ASPIRE is therefore 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.

The definition of ASPIRE middleware specifications has been a complex process that started from the review of state-of-the art collaborative and middleware platforms in deliverable D2.1. The reason for this review was to identify which middleware platform architectures were currently in use by both proprietary and open source developers, and, at the same time, to look into a new collaborative and innovation management process that would allow ASPIRE developers and stakeholders to collaborate with each other in a more efficient manner. The results in D2.1 pointed towards the reuse of open source components such as those from the Accada project (currently Fosstrak) and which is based on an EPC (Electronic Product Code) architecture. In addition to the review of state-of-the art, ASPIRE consortium organized an online questionnaire, RFID information days in different countries of the consortium and in general a market research regarding the requirements of SMEs in terms of RFID applications, their IT infrastructure, their knowledge about RFID technology and in general their willingness to adopt the new RFID paradigm replacing old ones such as optical bar scanners. The results in D2.2 have indicated that SMEs do actually have high requirements on RFID solutions but they are reluctant to adopt it due to little knowledge of the technology, security and privacy issues, limited IT infrastructure at their premises and mainly due to the high costs of associated solutions. This requirement collection process was later reflected in the ASPIRE middleware architecture specifications provided in deliverables D2.3 and D2.4.

RFID middleware platforms consist of generic building blocks, for example filtering, aggregation and collection functionalities, which refer, respectively, to the elimination of those raw RFID readings that are not relevant for upper layer applications, the combination of readings from more than one antenna or more than one reader in order to compensate for reading errors, and the simple managing of all the received raw RFID readings. Other building blocks are the business event generation, which translates RFID readings into events with business meaning, information repository, which contains the true identity or information about an RFID identifier and the object name service which has the same role of a DNS (directory name service) in Internet architectures of providing the network address where the information about an RFID identifier is located. In summary, an EPC-extended architecture has been chosen with extra features that make it more attractive for SMEs, such as added value sensing solutions and reader anti-collision management, and that considerably simplifies and reduces the cost associated with the deployment of RFID solutions. EPC architecture was also chosen because it provides a good trade-off between low and high complexity hardware solutions, which is in line with the SMEs requirements of providing, at an initial stage, low complex hardware components by pushing the intelligence or complexity towards the middleware components. In this deliverable we provide implementation details for such middleware components, which are in line with the Aspire middleware architecture specifications. This deliverable reflects also previous deliverables in work-package 3, mainly D3.2 and D3.2 which dealt, respectively, with tag data translations and the filtering and collection middleware server.

In this deliverable we will provide information regarding the development of the core Aspire middleware components. Namely,

- Section 4 will provide a detailed description of the Aspire middleware architecture
- Section 5 will discuss the environment on which the Aspire core middleware components will work.
- Section 6 will provide details of the tag data translation component.
- Section 7 will provide implementation details on the components that will handle the readers and sensors
- Section 8 will provide development details of the filtering and collection component, which is responsible for the filtering of the raw data
- Section 9 will provide details of the business event generator component that handles higher level data reports and add business logic.
- Section 10 will provide details on the EPCIS (Electronic Product Code Information Services) components which is responsible for the storage and provision of the processed information.
- Section 11 will give initial details on the ONS (Object Name Services) component which acts as a name service in the network of things that the middleware establishes.
- Section 12 will provide information on the context analysis component which is responsible for handling the actuators and providing input on the rest of the core components based on information gathered from a number of sources.
- Section 13 will provide information regarding the connector component which is responsible for connecting the middleware with the actual consumers of the information: the legacy IT systems.

Finally we provide appendices with low-level information for developers.

Section 4 Overview of core ASPIRE middleware infrastructure

As it has been previously stated in deliverable D2.4, which defines the specifications on which the Aspire Middleware is structured on, 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 development. 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 (Integrated Development Environment) and tools are designed and intended to work on. It is responsible for handling the

- 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 architecture of the middleware is depicted in the following Figure 1.

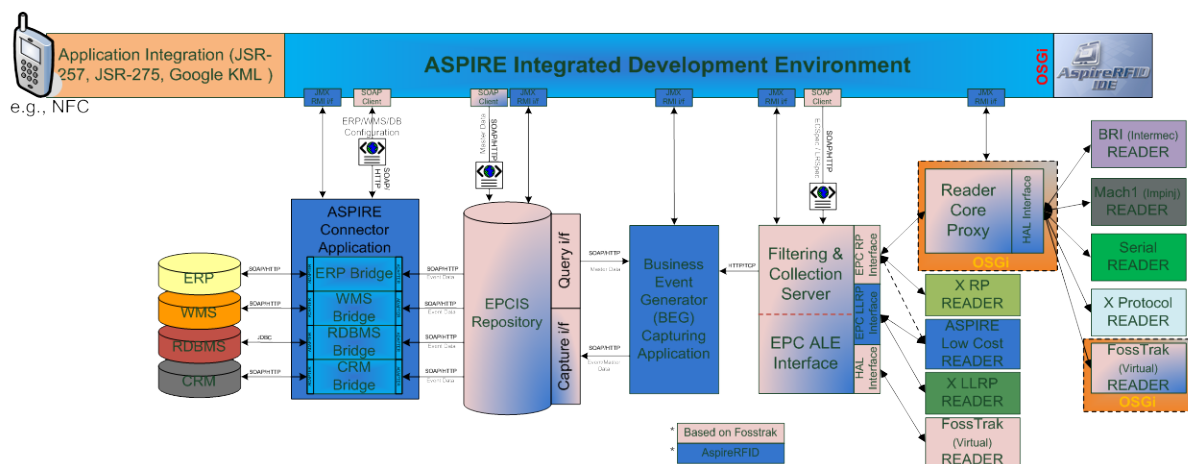


Figure 1 Aspire middleware architecture

Each node in this figure 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 (Application interface). At a higher level the hardware abstraction layer (HAL) is introduced which allows handling a heterogeneous landscape of readers by hiding 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.

The low level events, though lesser than the raw RFID reads, are significant in amount and do not provide high level – or business level – information. The role of this additional filtering and business events layer is handled by the Filtering and Collection (F&C) and 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 server.

This information (i.e. business events) is then forwarded to a higher hierarchical level, where it is consumed by the Information System level (IS). The IS level 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., Warehouse Management Systems (WMS), Enterprise Resource Planning (ERP) systems and corporate databases). 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. electronic data interchange (EDI), extensible markup language (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 this functional plane in the architecture, there is a management plane which, as the name suggests, manages and orchestrates the subsequent components. The management plane ensures that the middleware components comprising an ASPIRE system operate appropriately, while at the same time providing a functionality for runtime management of the modules (e.g., starting, stopping, deploying and (re)configuring components).

Under this prism, the following chapters drill down to the various architectural nodes and specify their functionalities, interfaces and current implementation statuses.

Section 5 Infrastructure components

5.1 OSGi (formerly known as Open Service Gateway Initiative)

5.1.1 Overview and purpose

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 (JVM). Although it is an effort that started in 1999, it was only recently that the software industry has shown a trend (e.g. Eclipse, Jonas, Glassfish) 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 shutdown 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. In the context of ASPIRE, the OSGi components will greatly simplify the management and deployment of its current and future modular components

5.1.2 Interfaces to other components

In the RFIDSuite (one of the branches of middleware within the ASPIRE project) the interface to other components are exposed as services registered in the OSGi service registry. Service dependency management is done via the Apache Felix iPOJO [15] component model. Communication between different OSGi applications (edge and premise) on the RFID Suite has been done using a message based middleware (Java Message Service) as well as Web Services and SMTP (Simple Mail Transfer Protocol).

5.1.3 Current implementation status

The edge and premise servers of the RFID Suite have been implemented using OSGi technology. As this branch is a continuation of an experiment originated from UFJ (University Joseph Fourier), it is necessary to evaluate which concepts can be brought to the main AspireRfid branch. Currently, only the Reader Core proxy is implemented as an OSGi application.

5.1.4 Future steps

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.

5.2 Application server

5.2.1 Overview and purpose

The Java Application Server (JAS) has been used in the RFID Suite for implementing the EPCIS and the ONS which also has some Discovery Services functionality embedded. This ONS was developed using a Web Services approach. 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.

5.2.2 Interfaces to other components

The EPCIS has functionality exposed as HTTP (Hypertext transfer text) 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.

5.2.3 Current implementation status

The current implementations are maintained by UJF in the RFID Suite branch of the OW2 SVN (subversion) repository.

5.2.4 Future steps

Parts of the RFID Suite shall be integrated with the main AspireRfid branch. The ONS approach that uses Web services is an interesting add-on to the standard DNS (Domain Name Service) 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.

5.3 Java Management Extensions (JMX)

5.3.1 Overview and purpose

Java Management Extensions are 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.

5.3.2 Interfaces to other components

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.

5.3.3 Current implementation status

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.

5.3.4 Future steps

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. The interfaces will be also compatible with the RFC-139 (JMX Control of OSGi) which is currently not yet available to the public.

Section 6 Tag Data Translation (TDT)

6.1 Overview and purpose

The EPC (Electronic Product Code) Network defines standards going from tag data to Application Level Event (ALE) in its architecture framework. This ALE is used by clients to obtain EPC data from sources.

At the lowest level is the Tag Data Standard (TDS) defined by EPCglobal. This data is retrieved thanks to a low-level reader protocol (LLRP) which communicates 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, read data from tags can be used with ALE (Application level event) , EPCIS, legacy applications, ONS or for writing information in tags. Nevertheless, the product code must be translated into different formats to be used in each of these scenarios. The tag data translation performs this task.

The tag-encoding URI (Uniform Resource Identifier) representation of an EPC is used with the Application Level Event. 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 translation between all these formats can be performed at any level of the architecture, so the TDT is a very important tool in the EPCglobal Network.

The ASPIRE TDT [1] provides a way to convert not only EPC tags, but also other identification standards such as GS1 bar codes, ISO (International Standard Organization) 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	11110000011000100000000000000000 000000000000000000011001011100000101

Figure 2 ISO 15693 Tag Data representation

6.2 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".

6.3 Current implementation status

The ASPIRE TDT is available in its 0.3 version. In this version, GS1 bar code, EPC tags, and ISO 14443 and 15693 can be translated into various formats.

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 or ISO); (iii) GS1 SI and code length (for output format GS1_AI_ENCODING); (iv) tag length, company prefix length and filter (for some input format). Figure 4 shows the TDT behavior. We use the Fosstrak implementation of the EPC TDT as core of EPC translation.

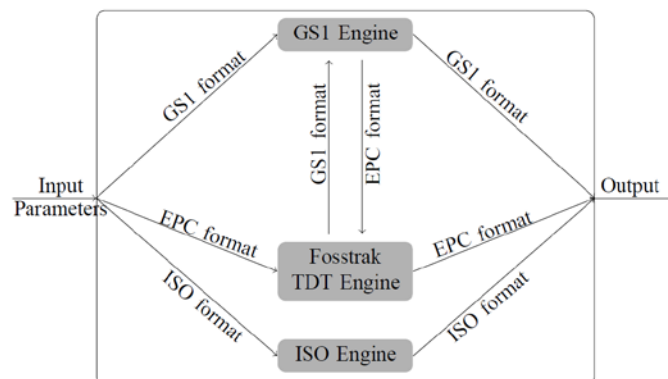


Figure 3: ASPIRE TDT Engine

Technical Requirements:

JDK	1.6.0_05 or higher
Memory	No minimum requirements.
Operating system	No minimum requirements. Tested on Linux

6.4 Summary and future work

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

Tag Data Standards	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
GTIN	Yes
GLN	Yes
GRAI	Yes
GIAI	Yes
GLN	Yes
ISO Tags	To be supported
14443	Yes
15693	Yes
15962	To be supported
uCode Tags	To be supported
Others IDs	To be supported
Mac address for Bluetooth and zigbee sensors	To be supported
Phone number (with country prefix) for cell phones	To be supported
Device id for OneWire devices (iButton)	To be supported

Table 1 Summary of types of tag supported by the TDT

Section 7 Reader interfaces

7.1 Low Level Reader Protocol (LLRP) interface

7.1.1 Overview and purpose

The LLRP protocol enables the Aspire middleware to communicate with LLRP protocol compliant readers. The low level reader protocol [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. In comparison to the previous reader protocol (RP), which is a high level protocol isolated from the physical and other low layer parameters, LLRP was designed after communities realized that it was convenient for the reader operation to have knowledge of low level parameters such as reader transmit power, frequency band configuration, etc in order to tackle in a more efficient manner problems such as reader collision and interference. LLRP also allows retrieval of manufacturer information of the reader and discovery of reader capabilities, among several other enhancements over the RP protocol.

7.1.2 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.

7.1.3 Current implementation status

This interface has been fully integrated into the Aspire middleware through the use of the open source 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].

7.2 Supported Readers

7.2.1 Overview and purpose

UJF has already develop drivers for several RFID readers such as TagSys Medio L100, TI Tiris 6350, ACR 122 (ie Touchatag), Violet Mirror in the RFID Suite 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 (ie legacy software in Java or other languages such as C# or C) and Bluetooth clients (ie NFC phones).

7.2.2 Interfaces to other components

In the RFIDSuite, 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 (ECA-event condition action- rule engine for instance).

7.2.3 Current implementation status

The current implementations relies on the Event Admin service as described previously

7.2.4 Future steps

The future steps are the porting of the existing drivers in order to be compliant with the EPCGlobal Reader Protocol (RP) standard version 1.1.

7.3 Near Field Communications (NFC)

7.3.1 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 has already met the mass market in Japan (50 millions NFC-enabled phones and 7 millions 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-Java Specification Request- 257) [13]. This API is mainly led by Nokia and enables to develop J2ME (Java 2 micro-edition) applications using NFC tags.

7.3.2 Interfaces to other components

NFC phones should be supported by the Aspire middleware as mass-market RFID readers. Moreover, NFC phones could query the ONS provided by Aspire using Web services or RESTful services.

7.3.3 Current implementation status

The RFID Suite 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.

7.3.4 Future steps

The future steps are the integration of the NFC library and the Bluetooth bridge as readers compliant with the EPCGlobal Reader Protocol standard version 1.1. The library will be completed by utilities components querying the future version of the Aspire ONS.

Section 8 Filtering and Collection (F&C)

8.1 Infrastructure

8.1.1 Overview and purpose

The filtering and collection (F&C) 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.

8.1.2 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.

8.1.3 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 (Lesser Public General License). The Aspire F&C consortium has worked and implemented changes and bug fixes on the original Accada middleware.

8.1.4 Future steps

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.

8.2 Anti-collision protocol

8.2.1 Overview and purpose

When a RFID tag enters the field of several readers, it can be read by all of them at the same time. To illustrate such a case, let's have a look at Figure 4. The tag enters the grey area which is the range of reading of readers 1, 4 and 5 and will be read uselessly twice. This wastes energy and data processing since requiring some tag filtering at the ALE level to identify such cases. Such a filtering required a fine spatio-temporal analysis of readers and their range of detection. Moreover, this supposes that each tag identifier is leveraged to the ALE level. This useless traffic wastes bandwidth and ALE resources. Since such a situation implies that an area is covered by at least two operational readers, data may be deteriorated because of collision among readers.

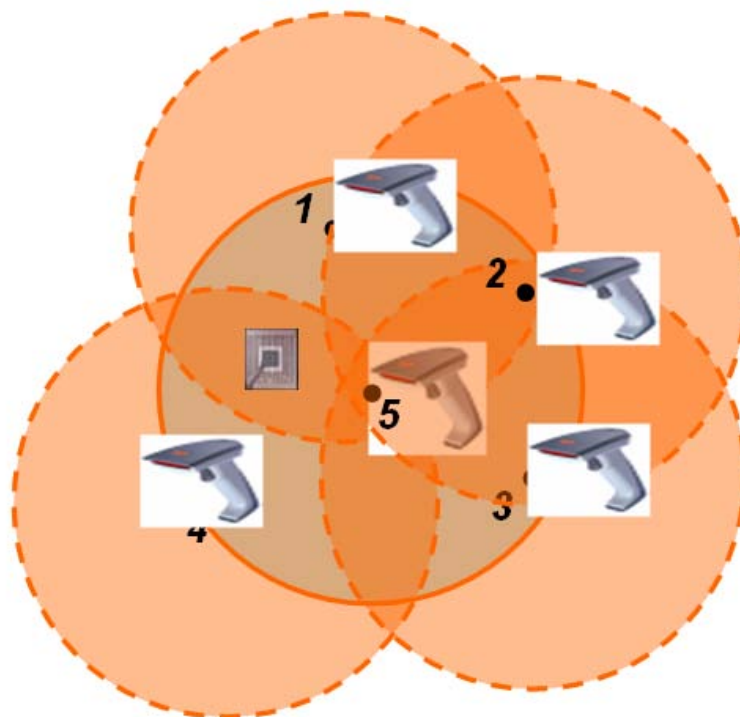


Figure 4 Example of reader collision

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 4, 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.

8.2.2 Current investigation status

We have investigated several trails and decided that one of the best way was to adapt some solutions from the sensor network research area such as in [1], [3], [4], [5]. Issues are pretty

similar and solutions for sensor networks have already been tested. Readers can be seen as sensors and with two readers being defined as neighbours if and only if they cover a common area.

Adaptations have to be run concerning the coverage area. In sensor networks applications, the coverage area is a disc with predefined radius. Intersections are thus easy to compute. In reader management, a simple algorithm has to be deployed to allow readers to know the area they cover.

Anyway, applying such a solution allows storing only a small amount of data on each reader and assailing them with lightweight algorithms. This will allow saving overall energy since some readers would be able asleep leading to less data being processed at higher levels.

8.2.3 *Future steps*

Future steps are to analyze deeply how far activity scheduling for sensor networks can be applied to reader management issues and to perform the adaptations.

Section 9 Business Event Generator (BEG)

9.1 Overview and purpose

The architecture introduces a Business Event Generator (BEG) module between the F&C and Information Sharing (e.g., EPC-IS) modules as shown in Figure 5 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.

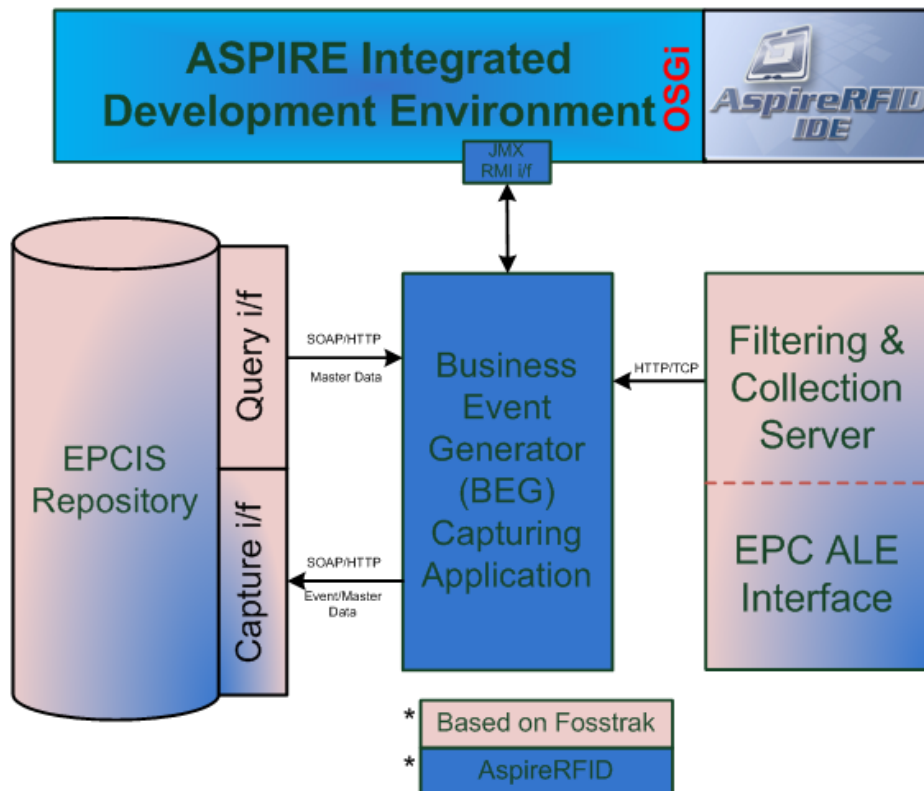


Figure 5 BEG interfaces

With the help of AspireRfid IDE (Integrated Data Environment) by describing the company's business processes and its underlying business infrastructure the required business events that constitutes 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. The latter events are submitted to the EPC-IS Repository, based on an EPC-IS capture interface and related bindings (e.g., HyperText Transfer Protocol (HTTP)/ Java Messaging Service (JMS)). The specification of the BEG is a valuable addition over existing RFID middleware architectures and platforms.

9.2 Interfaces to other components

The Business Event Generator Module interface provides five methods for interaction with the BEG Client which are all communicating with the BEG client by exchanging SOAP (Simple Object Access Protocol) messages.

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 Business Event Generator module is already serving. So with the help of this method someone can observe at real time the incoming IDs as they are reported to the BEG and are related with a specific transaction Event.

The second method is the “stopBegForEven” (boolean stopBegForEven(String eventID)) which is used from the BEG client to stop serving an already defined Event by sending to it its specific “EventID”.

The third method is the “getStartedEvents” (List<String> getStartedEvents()) Which returns a list of Event IDs that the BEG is serving.

The fourth method is the “startBegForEvent” (boolean startBegForEvent(VocabularyElementType vocabularyElementType, String repositoryCaptureURL, String begListeningPort)) which is used to set up the Business Event Generator module 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 ECRports 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”.

And 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.

9.3 Current implementation status

The technical status of the Business Event Generator module is that for its deployment, it is packed as a War file which is deployed on top of Apache tomcat 6.0 (and higher) and uses Java version 1.6 (and higher).

For implementing the required web services for its configuration and management interface it uses JaxWS (Java Webservices) framework and CXF/Spring technologies for its implementation.

For capturing the produced ECRports 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 also 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

Finally, the Event Types that the Business Event Generation module supports are the standard Event types defined in the EPCIS 1.1 Specifications which are:

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

9.4 Future steps

In addition to the already implemented features in the near future the Business Event Generator module is planned to be augmented with some new ones and maybe to be subjected to some changes even at its current Design.

Some of the possible changes/additions will be the creation of an interface for connecting/supporting

- Actuators that will be able to connect to the AspireRFID architecture through the BEG module 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 module will get implemented as an OSGi bundle that can be deployed on top of a JOnAS Application Server (open-source implementation of the Java EE 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/Access mechanisms for permitting or denying management and configuration access for its various interfaces.

Section 10 Electronic Product Code Information Services (EPCIS)

10.1 Overview and purpose

The ASPIRE Information Sharing repository and services are based on the EPCIS specification [10] and their components are in charge of:

- Receiving application-agnostic RFID data from the filtering & collection middleware through the Business Event Generation (BEG) application.
- Translating RFID data into corresponding business events. These events carry the business context as well (e.g., they refer to particular companies, business locations, business processes etc.).
- Making business events available and accessible to other upstream applications.

The Information Services of the ASPIRE Information Sharing middleware itself 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 in comparison to the underlying data capture and filtering & collection middleware modules.

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.

10.2 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 6. 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.

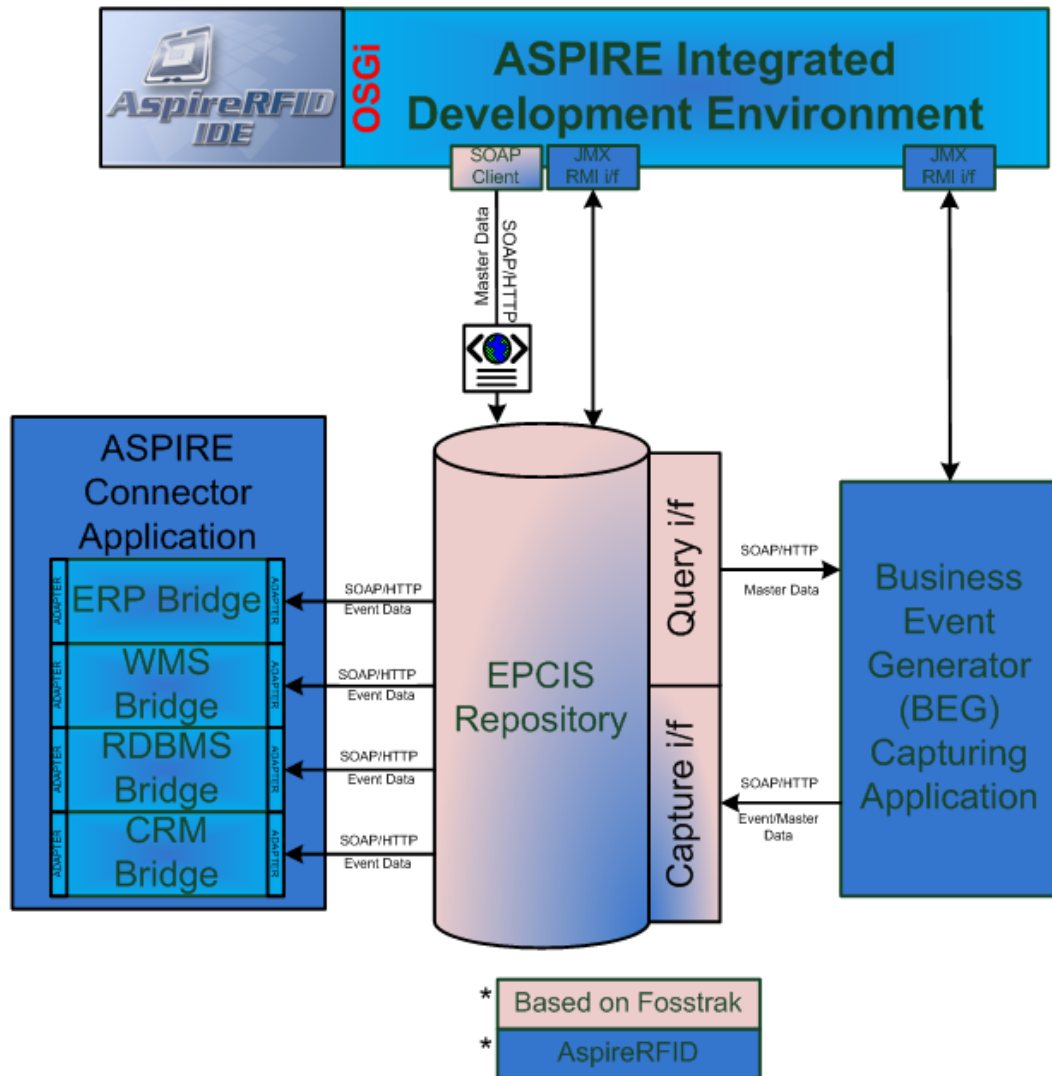


Figure 6 EPCIS interfaces

Please 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.

10.2.1 Master Data Capture Interface

A new feature that ASPIRE architecture introduces is the Master Data Capture Interface as shown in Figure 6 which enables the EPCIS module 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

10.3 Future steps

Except the already implemented functionalities of the Information Sharing repository there are still some that need to get implemented which are:

- Add new Field extensions for the existing Event Types extensions in the Data Definition Layer (Useful to store e.g. GPS-global positioning system- coordinates, temperature, hygrometry, shock events, survey answers, security check, etc.).
- Investigation of Implementing EPCIS repository as an OSGi bundle that can be deployed in JOnAS Application Server.
- Implement EPCIS specifications, and probably extended, Authentication/Access mechanisms.

Section 11 Object Name Service (ONS)

11.1 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 [EPC] (see Figure 5-1) [11]. This URI conversion is done in the local server by the TDT module. The figure 10-1 represents the ONS Architecture.

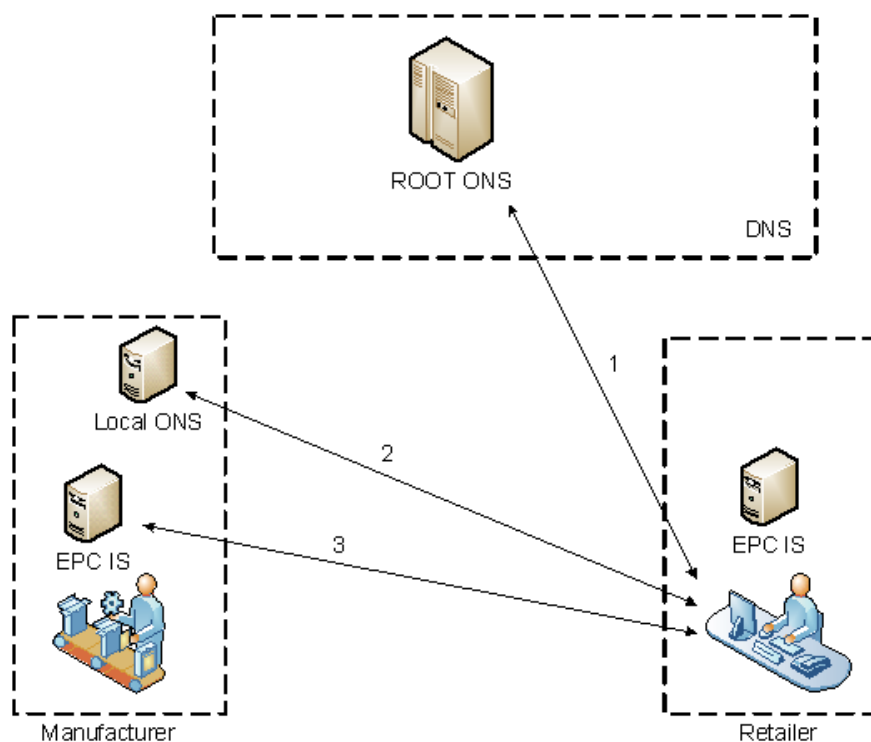


Figure 7 ONS Architecture

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).

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.

11.2 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 5). It communicates its results to the Business Event Generator and the EPC IS repository, where the information related to the product is recorded.

11.3 Current implementation status

A first version of the ONS component has been developed by UJF in the rfidsuite. It has been freely inspired by the model used by the EPC Global principles but this ONS can be queried using Web-based Services technologies instead of the DNS-based protocol. This solution is well suited for ASPIRE that does not want to get bind by a proprietary service. The development of the ONS component is ongoing and will soon be integrated in the AspireRFID Middleware.

Section 12 Context analysis

12.1 Overview and purpose

On top of RFID programmability, the ASPIRE RFID middleware platform has been designed to incorporate intelligence, enabling context-analysis and reasoning over numerous sensors observations. Reasoning enables the ASPIRE middleware to alleviate problems associated with the physical layer of the RFID network (e.g., incorrect readings). Furthermore, the ASPIRE middleware intelligence is also designed to support automated adaptation to the capabilities of the underlying readers network. Having the intelligence within the middleware platform will push functionality at the edge of the reader network and will allow the use of low-cost hardware (e.g., interrogators, gateways, tags) for the RFID solutions. A low cost reader is also part of the ASPIRE project and it is being developed in the context of WP5.

The context analysis component of the Aspire middleware platform is based on a powerful rule engine that constantly evaluates numerous operational parameters in semi real time and produces decisions based on the rules that have been predefined.

12.2 Interfaces to other components

The context analysis component is intended to receive input from the various Aspire middleware blocks and communicate its decisions to any interested component. The components that can provide input to the context analysis component are the following:

- The filtering and collection (F&C) server
This server will provide input to the component that is related to the filtered reading of RFID and sensor data, like temperature, pressure, etc. The context analysis component will be able to receive such data through the standard interfaces that the ALE standard defines which are also supported by the F&C server.
- The business event generator (BEG)
The BEG component is responsible for generating intelligent business events based on its configuration and input from the F&C server component. The context analysis component will be able to receive this kind of intelligent input and process it as long as there are existent rules defined in the engine. The component will be able to receive BEG data through the standard interface that this component defines.
- The EPCIS server
The EPCIS component is responsible, among others, for handling the business events subscriptions and for storing the events in its internal repository. The context analysis component will be able to subscribe to EPCIS events through the standard EPCIS interfaces and generate intelligent decisions which can be usable by many other components like the actuators, the connectors, etc.
- The connector component
The context analysis component will be able to receive data from legacy IT applications through the connector component and combine this information with other sources to make intelligent decisions.

In addition to getting input from the aforementioned sources, the context analysis component will be able to provide input, after asserting and evaluating the input data, to the following components:

- The actuator components
As these components are responsible of generating reactions to certain triggers, they can be defined as the primary consumers of the context analysis products (decisions).
- The Connector component
The legacy IT applications that will be connected with the Aspire middleware platform through the connector component may be able or may require additional input, apart from the data coming from the EPCIS repository. This input may be relevant to the general operational status of the infrastructure, etc. Thus the context analysis component will be able to provide such input through the connector component when requested to do so.

12.3 Current implementation status

The development of the context analysis component is a relatively complex task that requires other components from which requires input or which require input from this component to be concretely designed (e.g. actuators). Thus, the context analysis component is currently in a premature state.

The Aspire consortium responsible partners have currently defined the operating environment of the component's rule engine. For this purpose we have agreed on the use of the Jess rule engine. **Jess** is a rule engine and scripting environment written entirely in Sun's Java language and uses an enhanced version of the **Rete** algorithm to process rules. It is available in a dual licence scheme for academic and commercial use.

12.4 Future steps

The future steps of this component can identified in the process of **defining a set of rules** that will analyze the context and produce the intelligent decisions, in **coding** the actual rule engine and finally, **integrating** the component with the rest of the Aspire middleware components.

Section 13 Connector

13.1 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 an 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.

13.2 Current implementation status

The connector component is intended to provide means of transparency between an application and the EPCIS repository. An EPCIS repository, as stated in a previous chapter, is able to collect business level events, and provide this information using either a push or a pull concept. The Connector that we define in this paper 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.

13.2.1 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 []. 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.

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 1 is a description for each of the fields.

```
<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>
```

13.2.2 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 through 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 []. 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>
```

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 startObservingTransaction	Use with stopObservingTransaction	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 <i>1,31</i> 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. Error! Reference source not found.	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.

Contract: 215417
Deliverable report – WP3 / D3.4a

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 1 Subscription parameters field's explanation

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

13.3 Future steps

The connector concept needs to be validated in real world situations. Such situations would involve widely adopted legacy IT systems being able to communicate with a 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
- The EPCIS to be able to send business events to registered

In order to be able to test the validity of the connector concept and design, we will need to develop connectors for major legacy IT systems and evaluate the operational behavior in real cases. 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.

Section 14 Evolution of Aspire middleware infrastructure (D3.4b)

In the previous chapters we have the work done until project month 18 regarding the core Aspire middleware infrastructure. A lot of work needs to be done until the final version of this deliverable so the realistic deployments of the middleware can be possible. The most immediate action is the merge of the development branched of the project development forge (available at <http://forge.ow2.org/projects/aspire/>). The two branches (AITdev and rfidsuite) are the results of effort done in parallel and lead by two Aspire partners. Here is the list of the components that each branch is going to provide:

- Provided by AITdev branch
 - Reader Core Proxy
 - ALE server
 - EPCIS
 - BEG
 - Connectors
- Provided by rfidsuite 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 have recently been arose, the consortium will need to reevaluate the option of using licensed software by Fosstrak and will investigate the possibility of using exclusively consortium developed components. In this case some of the core components of the Aspire middleware infrastructure will be replaced by new implementation with possibly new configurations.

A final step in the evolution of the core infrastructure components will be the development of the components that haven't yet entered the development phase but are currently frozen in the analysis phase.

Section 15 Conclusions

In this deliverable we have analyzed the core components of the Aspire middleware, providing information for each of these components until the time of delivery of this document. As it is evident by the interim nature of this deliverable, this analysis is by no means complete, but, on the other hand, is a work in progress. It intends to define the current implementation status of the middleware components along with the planned future steps that can currently be identified as important to complete the specified components' functionalities. Despite that, all work done can be evaluated as of high importance for the rest of the components that will be completed until the final version of this deliverable is due. That is because of the core nature of the components currently implemented.

Nevertheless, the currently developed software, while in its infancy for realistic deployments, is an actual infrastructure for community development, making available to the world open source RFID middleware components.

Moreover we need to draw focus on the need to change existing components that have been defined on this document, because of the licensing issues that have been identified in section 13. These components along with the development details of components not currently available will be the subject of final version of this deliverable (D3.4b).

Section 16 List of Figures

Figure 1 Aspire middleware architecture..... 13
Figure 2 ISO 15693 Tag Data representation 17
Figure 3: ASPIRE TDT Engine..... 18
Figure 4 Example of reader collision..... 23
Figure 5 BEG interfaces 25
Figure 6 EPCIS interfaces 29
Figure 7 ONS Architecture..... 31

Section 17 List of Tables

Table 1 Subscription parameters field's explanation..... 41

Section 18 References and bibliography

- [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 Theory and 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 -
<http://www.epcglobalinc.org/standards/llrp>
- [7] LLRP-Toolkit - <http://www.llrp.org/>
- [8] EPCglobal Application Level Events standard - <http://www.epcglobalinc.org/standards/ale>
- [9] Fosstrak project - <http://www.fosstrak.org/>
- [10] EPCglobal Electronic product Code Information Services (EPCIS) -
<http://www.epcglobalinc.org/standards/epcis>
- [11] EPCglobal Object Name Service (ONS) - <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>

Section 19 Appendix A – Filtering and Collection component (based on Fosstrak implementation) user guide

19.1 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.)

19.2 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 (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.

19.3 Configuration

This chapter will give a short overview to the configuration files available. These files allow you to adapt the aspireALE to your needs. You will 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 propertie-file is the main config for the ASPIRE ALE. You will 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 clients.

19.4 Logical Reader Configurations

This guide shall introduce 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.

19.4.1 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
- example

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

Dynamic Definition

- The xml must contain exactly one LRSpec definitions.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>  
<ns3:LRSpec xmlns:ns2="urn:epcglobal:ale:wsl: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:wsl: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:wsl: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.

Section 20 Appedix B – EPCIS (based on Fosstrak Implementation) users guide

20.1 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).

20.2 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, follow the step-by-step tutorial outlined below: 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. 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'll 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'll use the user name epcis and password epcis).

```
mysql> GRANT SELECT, INSERT, UPDATE, DELETE 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 : aspireRfidEpcisRepositoryWEB-INFclassesql 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.

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.

20.3 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 configuration files relevant to the user of the application: application.properties, context.xml, and log4j.properties

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

1. to configure the behaviour of the epcis-repository application
2. 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/EpcisRe
pository

1. maximum number of result rows allowed for a single query before a
2. QueryTooLarge exception is raised

maxQueryResultRows=1000

1. maximum time in milliseconds to wait for a query to finish before a
2. QueryTooComplex exception is raised

maxQueryExecutionTime=20000

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

insertMissingVoc=true

1. the schedule used to check for trigger conditions - the values provided here
2. are parsed into a query schedule which is used to periodically check whether
3. incoming events contain a specific trigger URI

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

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

dbResetAllowed=false
dbResetScript=/sql/reset_epcis_with_demo_data.sql

1. the location of the EPCglobal EPCIS schema

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

1. the location of the EPCglobal EPCIS MasterData schema(nkef)
```

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

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

trustAllCertificates=false

1. 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).

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.

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.

Section 21 Appendix C – Connector component Users Guide and Developer Guide

21.1 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.

21.2 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.

21.2.1 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.

21.2.2 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.