2007 - 2013

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

Data Collection, Filtering and Application Level Events

Due date of deliverable:	M15
Actual Submission date:	30/4/09

Deliverable ID:	**WP3/D3.3**
Deliverable Title:	Data Collection, Filtering and Application Level Events
Responsible partner:	AAU
Contributors:	Nikos Kefalakis (AIT) Nektarios Leontiadis (AIT) John Soldatos (AIT) Nathalie Mitton (INRIA) Loïc Schmidt (INRIA) David Simplot-Ryl (INRIA) Ramiro Robles (IT) Didier Donsez (UJF) Kiev Gama (UJF) Lionel Touseau (UJF) Sofyan M Yousaf (IT)
Estimated Indicative Person Months:	28

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

Revision: 0.11
Dissemination Level: PU

Document Information

Document Name:	**Data Collection, Filtering and Application Level Events**
Document ID:	**WP3/D3.3**
Revision:	**0.11**
Revision Date:	**29 April 2009**
Author:	**AAU**
Security:	**PU**

Approvals

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

Reviewers

Name	Organization	Date	Comments	Visa
Mathieu DAVID	AAU	26 Mar 09	Needs additional work and revisions	
Mathieu DAVID	AAU	15 Apr 09	Close to final version Consider comments	
Sofyan M Yousaf	OSI	29 Apr 09	Consider Comments	

Document history

Revision	Date	Modification	Authors
0.1	25 Feb 09	First draft	Nikos Kefalakis
0.2	20 Mar 09	Section 9, augmented Section 3	Nathalie Mitton
0.3	24 Mar 09	Section 1, Section 2, augmented Section 3	Ramiro Samano Robles
0.4	26 Mar 09	Augmented Section 9	Sofyan M. Yousuf
0.5	27 Mar 09	augmented Section 2	Ramiro Samano Robles
0.6	02 Apr 09	Appendix I and II, Section 6, Section 7,	Nikos Kefalakis

		augmented Section 2	
0.7	09 Apr 09	Section 10	Ramiro Samano Robles
0.8	20 Apr 09	Insertion of ALE extensions on Appendix I	Didier Donsez and Kiev Gama
0.9	21 Apr 09	Harmonization of the document and Re-Ordering of some sections	Nikos Kefalakis, John Soldatos
0.10	29 Apr 09	Minor Corrections/Additions at Section 3, Section 4 and Section 7	Nikos Kefalakis
0.11	29 Apr 09	Section 8	Didier Donsez, Kiev Gama & Lionel Touseau

Content

ID: D3.3_Data Collection, Filtering and Application Level
Events.doc10
Revision: 0.10

Date: 29 Apr 2009

Security: Public
Page 4/58

ID: D3.3_Data Collection, Filtering and Application Level
Events.doc10
Revision: 0.10

Date: 29 Apr 2009

Security: Public
Page 5/58

Section 1   Executive Summary

This deliverable describes three crucial architectural functionalities and modules of the ASPIRE middleware platform: Data collection, filtering and application level events. The data collection functionality refers to the ability to capture and/or aggregate the data coming from different RFID reader(s), whereas filtering aims to select only that part of the information that is relevant for upper layers. Finally the application level event (ALE) functionality provides the actual translation from a low level RFID event into a higher level language relevant to upper layers. The present deliverable accompanies the prototype implementation of these functionalities as part of the F&C server of the AspireRfid project. The prototype implementation is available for download at http://wiki.aspire.ow2.org/, while it is also provided as part of CD-ROM that comes with this deliverable. Note the core part of the deliverable is the prototype implementation, while the present report can be seen as a complementary documentary report. Additional documentation is part of the project's Wiki and forge.

After going through this deliverable the reader will have a detailed picture of the aforementioned functionalities, the status of implementation of their different features in the ASPIRE architecture, the modules/servers that host them and the interfaces with the requesting clients or upper layer modules. Hence, this report has a dual role: (a) for end-users and integrator it presents and illustrates (based on examples) the list of features that they can use as part of the F&C implementation and (b) for developers and open source contributors it reveals the gaps that need to be filled based on additional contributions. It is worth mentioning that although the implementation was made following an EPC (Electronic Product Code)-like architecture, ASPIRE middleware is potentially applicable to other non-standardized readers thanks to a Hardware abstraction layer (HAL) which is able to translate their protocols into ASPIRE semantics.

ASPIRE is developing an innovative royalty free middleware platform. This middleware platform is the primary target of the open source "AspireRfid" project (http://wiki.aspire.ow2.org/), which has been recently established in the scope of the OW2 community. In the scope of large scale deployments, RFID systems generate an enormous number of object reads. Filtering those reads is a key prerequisite to implementing non-trivial RFID applications. This is because RFID applications need to concentrate on and receive certain application level events (e.g., the appearance of an object, the disappearance of an object), while at the same time ignoring reads that 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 as possible in the architecture.
- Minimize the amount of "business logic" embedded in the Tags.

The Filtering and Collection (F&C) Middleware is intended to facilitate these objectives by providing a flexible interface (ALE (Application Level Events) 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 (see Middleware Building Blocks and ASPIRE applications) 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 architecture (described in Deliverable D2.3), 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.

The ASPIRE (F&C) Middleware, (which is part of the AspireRfid Open Source Project, available at http://wiki.aspire.ow2.org/) has been implemented based on the licensing of the FossTrak EPC-ALE implementation under the LGPL license. On top of this licensed middleware ASPIRE partners have implemented a number of extensions, customizations and bug fixes. The later extensions involve the implementation of LLRP (Low Level Reader Protocol) connectors for bridging the F&C middleware with LLRP compliant readers, the addition of sensor extensions, as well as the porting of the middleware to lightweight OSGi (Open Service Gateway Interface) containers with a view to enabling modular and fine-grained control over the F&C servers. These extensions and customizations render the F&C middleware compliant to the ASPIRE architecture. In addition, ASPIRE has implemented a tool facility enabling the flexible configuration of the F&C server. The above extensions and tools are detailed in this deliverable. For completeness reasons, F&C features supported as part of the FossTrak/Accada implementation are also included in an Appendix.

Section 2   Introduction

Despite the simplicity of the operational principles of RFID technology (i.e. tags responding to readers requests), the design of a complete RFID system encompasses complex interactions not only between different layers of the OSI (Open Systems Interconnection) model, but it also involves several market, privacy, security, and business issues. This heterogeneous landscape calls for a middleware platform which is able to consider all these complex variables in a flexible and modular way, which is able to provide a starting point for future upgrades and innovations, and which considerably reduces the implementation costs of RFID solutions.

The main goal of ASPIRE is to develop a lightweight, innovative, royalty free, open source and programmable RFID middleware platform that will facilitate European companies in general and SMEs in particular to develop, deploy and evolve RFID solutions. In-line with its open-source nature this platform aims at offering immense flexibility and maximum freedom to potential developers and deployers of RFID solutions. This versatility includes the freedom of choice associated with the RFID hardware (notably tags and interrogators) which will support the solution. The ASPIRE middleware solution is developed as part of the AspireRfid Open Source Project, which is available for download at: http://wiki.aspire.ow2.org/.

This deliverable is concerned with three of the main functionalities of the ASPIRE middleware platform: Data collection, filtering and application level events. It also describes the modules/servers that implement such functionalities and the interfaces between such modules and clients or upper layer applications that are subscribed to their services. However, before going into the details of software implementation this section enlists the main components and functionalities of a generic RFID system according to the work in [10] where the authors make an exhaustive analysis of RFID interfaces and functionalities using a non-EPC framework. The objectives of this section are the following: to identify the functionalities tackled by this deliverable within an entire RFID system, to devise which components or hardware modules may host them, to map them into the EPC-architecture and set of standards and finally relate them to the ASPIRE architecture defined in previous documents. The section concludes with a description of the contents of this deliverable and its organization.

Note that the core of Deliverable D3.3 is the prototype implementation of the AspireRfid F&C server. The present report aims at accompany this deliverable, serving as complementary documentation. Please note that additional documentation is available at the AspireRfid Wiki portion of the F&C server at: http://wiki.aspire.ow2.org/.

An RFID system consists of tags, readers, controller appliances and application servers. The main functionalities of an RFID system can then be distributed among these hardware modules depending on the architecture selected or

application targeted. According to [10] the functionalities of an RFID system are divided in four categories enlisted below:

1. Base service set (BSS))"Over the air"

- *Transponder singulation. Collects the identification numbers (ID) of (selected) transponders in range.*
- *Transponder ID programming.* Writes identification numbers to transponders.
- *Transponder Memory Access.* Reads from and writes to the general purpose memory on a transponder.
- *Transponder Deactivation.* Disables the transponder for privacy reasons.

2. Configuration service set (CSS)
- *Network Interface configuration.* Discovers and sets reader networking parameters and identity, e.g. the IP address.
- *Firmware management.* Distribute and manage firmware version on readers
- *Antenna, Tag population and memory selection.* Specify reader antennas and tag population to be inventoried. In case of tag memory access, specifies memory fields to be accessed.
- *Base Service set scheduling.* Sets how different BSS services, such as tag inventory, access, and deactivation, are triggered and stopped.
- *RF transmitter configuration.* Sets transmit channel, hop sequence, and transmit power for readers.
- *Air Interface Protocol Configuration.* Configures timing, coding and modulation parameter of a specific air interface protocol on the readers.

3. Monitoring service set (MSS)
- *Network connection monitoring.* Check that the reader can communicate captured RFID data over the network.
- *RF environment monitoring.* Check RF noise and interference levels to safeguard reliable identification operation.
- *Reader Monitoring.* Check that the reader is up and running and executing BSS as configured for example via monitoring the number of successful/-failed read and write operations.

4. Data processing service set (DPSS)
- *Filtering.* Removes unwanted tag identifiers from the set of tag identifiers captured.
- *Aggregation.* Computes aggregates in the time domain (entry/exit events) and the space domain (across reader antennas and readers) and generates the corresponding 'super'-events.
- *Identifier Translation.* Translates between different representations of the identifier.
- *Persistent storage.* Stores RFID data captured for future application requests.

- *Reliable messaging*. Allow RFID data to be delivered reliably in the presence of software component, system and network failures.
- *Location/Movement estimation*. Detects false positive reads of far-away tags that are outside the 'typical' read range and estimate the direction of movement.
- *Application Logic execution*. Interprets the RFID data captured in an application context and generate the corresponding application events

Some of the above functionalities are clearly hosted by specific hardware modules. For example, the base service set (BSS) of functionalities, which is related to the direct interaction with tags, is always hosted by the reader or interrogator. In comparison, configuration and monitoring services are always hosted by the controller appliance or server. Conversely, the data processing service functionalities can be distributed among any of the hardware modules depending on the selected RFID architecture. Although two main architectures can be clearly identified, one centralized and another decentralized, hybrid versions combining bits of both of them might also exist. The centralized and decentralized approaches are shown in Figure 1 and 2, respectively. These architectures also imply different degrees of complexity at the reader and at the central controller appliance. Since ASPIRE aims at reducing the processing complexity at the hardware reader level, it will mainly adopt a centralized architecture. This also implies that the filtering and collection functionalities will be mainly hosted in a controller appliance which is commonly implemented in a server or groups of servers. The aim of this filtering and collection functionality server is therefore to collect raw RFID data coming from different readers, filter those parts of information that are irrelevant to upper layers and to translate them into higher level semantics understandable to application servers or to other users of the F&C server.
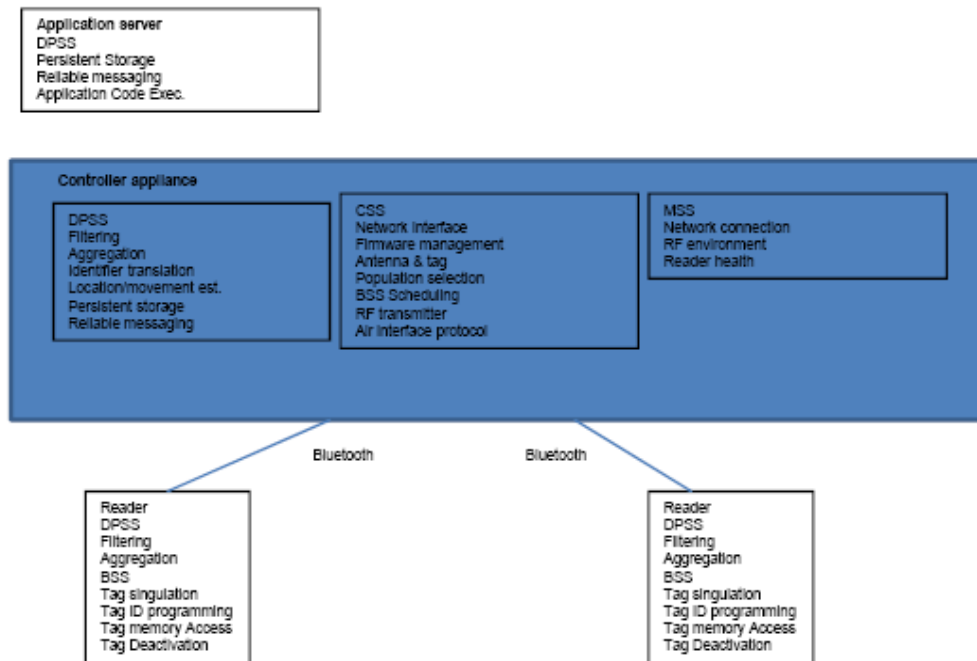
Figure 1 RFID system configuration with centralized architecture
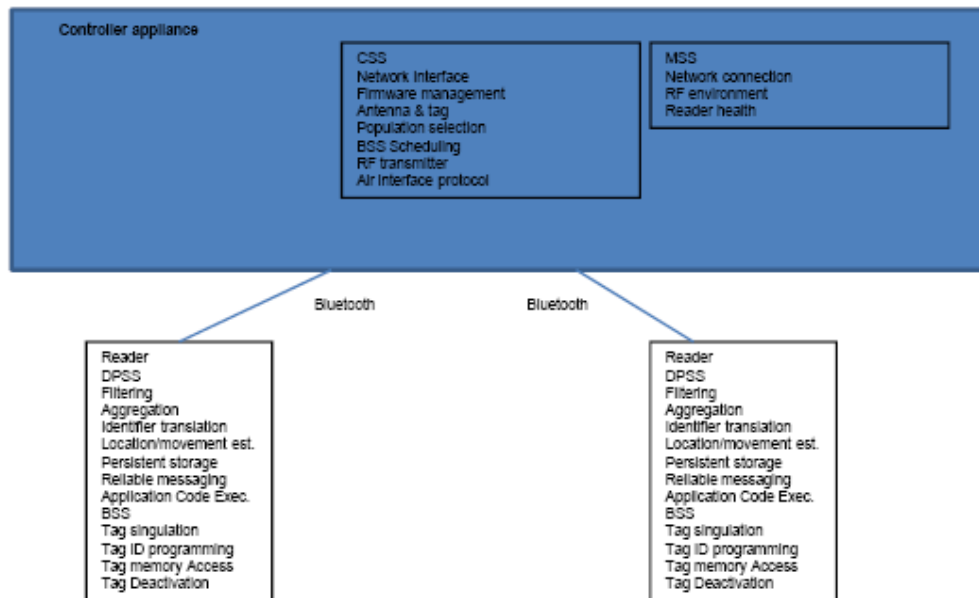
Figure 2 RFID system configuration with decentralized architecture

The EPC set of standards have been created with the aim of providing RFID application developers a common reference framework that will help in the fast adoption of RFID in current markets. The EPC standards consists of the definition of the radio interface between readers and tags (Gen 1 or Gen 2 UHF standards), reader protocols for the management and configuration of readers or

interrogators (RM –reader management-, Reader Protocol –RP- and Low Level Reader Protocol –LLRP-), and the standards for the interaction with upper layer procedures (Application Level Events –ALE- and Electronic Product Code Information Services –EPCIS-). The EPC network roles and functionalities can be observed in Figure 3, where data and device management functions are mainly comprised by the reader protocols, the filtering and collection server and the Application level event standard. In comparison, the data interpretation functionalities are mainly carried out by the EPCIS standard.

Figure 3 EPC Network roles and interfaces

Let us now relate the EPC network architecture and standards with the RFID functionalities previously explained. The following figure summarizes the functionalities that each EPC specification carries out.

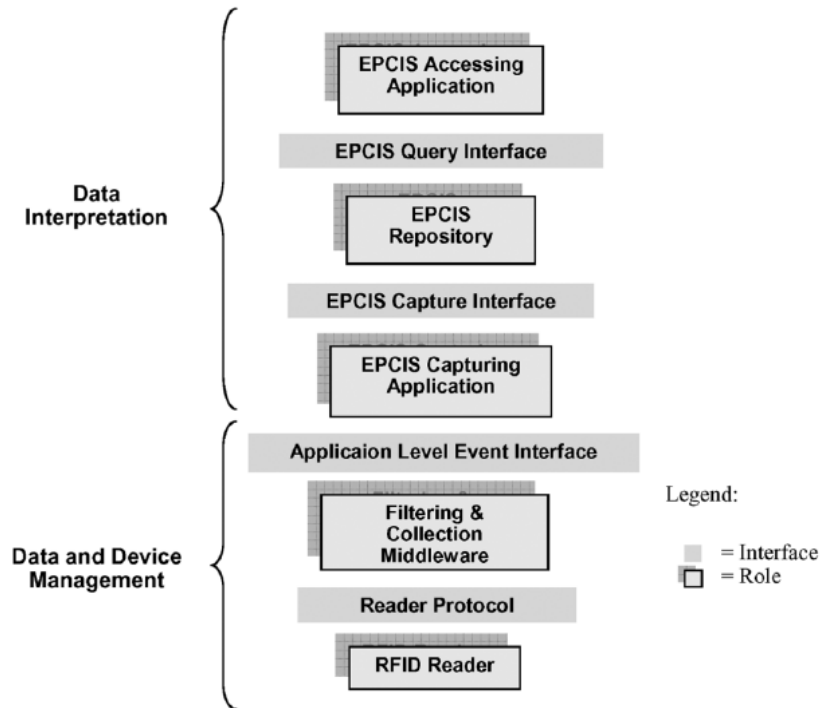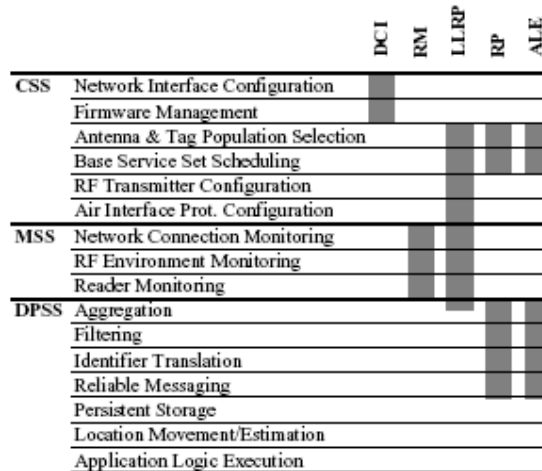|		DCI	RM	LLRP	RP	ALE
CSS	Network Interface Configuration	■				
	Firmware Management	■				
	Antenna & Tag Population Selection			■	■	■
	Base Service Set Scheduling			■	■	■
	RF Transmitter Configuration			■		
	Air Interface Prot. Configuration			■		
MSS	Network Connection Monitoring		■	■		
	RF Environment Monitoring		■	■		
	Reader Monitoring		■	■		
DPSS	Aggregation			■	■	■
	Filtering				■	■
	Identifier Translation				■	■
	Reliable Messaging				■	■
	Persistent Storage					
	Location Movement/Estimation					
	Application Logic Execution					

Figure 4 High-level overview of the services supported by the EPCglobal specifications [1O]

This figure shows that the main standards related to the filtering, collection and application level event are the LLRP (Low level reading protocol), RP (reader protocol) and ALE (application level event). Their approach to such functionalities is, however, different. Whilst LLRP and RP are mainly concerned with the interface between the middleware platform and the reader, the ALE specification is concerned with the interaction between the middleware and business related servers. LLRP is configuration, monitoring and some limited data processing functionalities. In comparison RP comprises limited configuration capabilities and better data processing capabilities. Finally, ALE also deals with configuration and data processing capabilities.

Since the ASPIRE architecture is mainly based on a centralized EPC architecture, the filtering and collection capabilities will be mainly concentrated in a dedicated server called ALE or F&C server, while some limited low level filtering capabilities will be left as an optional feature to deploy at the reader level. This server interacts with a heterogeneous reader landscape, thus being able to interpret LLRP, RP and HAL messages, with a business event generator through an HTTP/TCP interface and with the management platform through a SOAP/HTTP interface. Details are given in the main body of the deliverable.

It is worth to note that as codebase for developing the AspireRFID Filtering and Collection (F&C or ALE) module the OSS (Open Source Software) Fosstrak [1] was licensed and used which was extended and bug fixed to comply to the ASPIRE's Architecture Specifications. The present document focuses on the illustration of features, which have been entirely developed in the scope of AspireRfid, while the FossTrak implementation is detailed in an Appendix

This deliverable corresponds to the parts of the ASPIRE middleware infrastructure that will perform collection and filtering of tag streams, along with constructions of application level events. The present document is the report, whereas the Implementation and Documentation can be found at the AspireRFID Forge (http://forge.ow2.org/projects/aspire) and Wiki

(http://wiki.aspire.ow2.org/) page respectively. The executable final version of the F&C implementation (aspireRfidALE[version].zip) and the ALE Server Configurator (AspireRfidIdeToolCollection[version]) can be found at the AspireRFID Forge Files page (http://forge.ow2.org/project/showfiles.php?group_id=324), the source code of the implementations can be found at the AspireRFID SVN (http://forge.ow2.org/plugins/scmsvn/index.php?group_id=324). Directions on how to use the AspireRFID F&C module and ALE Server Configurator can be found at the AspireRFID Wiki documentation page http://wiki.aspire.ow2.org/xwiki/bin/view/Main.Documentation.AspireIDE/AleServerConfigurator and http://wiki.aspire.ow2.org/xwiki/bin/view/Main.Documentation.AspireIDE/AleServerConfigurator respectively.

The structure of the contents of this deliverable is as follows: Section 3 presents the general scope of the ALE or F&C server in the context of the ASPIRE architecture and a description of its modes of operation. Section 4 presents the extensions and fixes carried out by AspireRfid over the FossTrak implementation (http://www.fosstrak.org), while Section 5 illustrates the ALE sensor extensions developed in the scope of AspireRfid. Section 6 describes the configuration tools of the servers, which have been implemented and are available as part of the AspireRfid project. Section 7 presents few complete implementation examples of the F&C functionally in order to allow the reader to fully understand the operation and the importance of the F&C server. Finally, Section 8 presents the results of some preliminary low level reading investigations and Section 9 draws conclusions.

Section 3   Scope of AspireRFID F&C Server

3.1   Application Level Events

Application Level Events (ALE) is a software interface through which client applications may interact with filtered, consolidated EPC data and related data from a variety of sources. In particular, ALE provides a convenient way for applications to read and write RFID tags, interacting with one or more RFID reader devices and is implemented at the AspireRFID filtering and collection module.

ALE provides an especially convenient programming model for application writers. Using ALE, an application makes a high-level description of what data it wants to read from or write to tags, over what period of time, and with what filtering to select particular tags. The ALE implementation then finds the most appropriate way to fulfill such requests, interacting with RFID readers or other devices as necessary. Application writers are shielded from details of device configuration and management, and may rely upon the ALE implementation to handle data conversions including those specified by EPC and ISO data standards. [16]

3.2   Data Collection

The ASPIRE F&C middleware module must represent a single interface to the potentially large number of readers that make up an RFID system deployment. This allows applications to subscribe to a particular pre-defined specification, which is then used along with the Logical Reader (LR) definition to configure the corresponding reader devices using the underlying reader access mechanisms. Once the readers capture relevant tag data they notify the middleware which combines and aggregates the data arriving from different readers in a report that is sent according to a pre-determined schedule to the subscribed applications.

3.3   Filtering

Since the middleware receives data from multiple readers, it provides specific filtering functionality depending on the different pre-defined specifications. So redundant events from different readers observing the same location are not included to the dispatched report accomplishing the reduction of filtering and aggregation required for the registered application to interpret the captured RFID data.

3.4   F&C Interfaces

The ASPIRE F&C middleware must implement one interface to communicate with the different types of readers and two other interfaces to communicate with upstream layers (i.e. BEG (Business Event Generation) module and ALE Server Configurator as shown in Figure 5). In particular:

- One interface is needed for transporting the RFID data. The TCP/HTTP protocol therefore is adopted to this end. The transport interface must enable BEG and/or applications to receive RFID data either in a "push" or in a "pull" fashion (either by subscribing for or by requesting a report). This interface will be important for the support of the information flow of RFID data from tags and readers to the business repository.

- A second interface (based on the SOAP and/or other XML messaging protocols) for managing the F&C server and controlling its operations. This interface will not target the transport of RFID readings. Rather it will allow definition of reading specifications, subscription of clients to the results of particular filtering specification, as well as definition and management of logical readers. This interface will be used by all the ASPIRE management and development tools, which will need to configure and/or program the F&C server operation.

The figure below shows the ASPIRE middleware architecture and the interactions of the filtering and collection server with the other modules of the architecture.
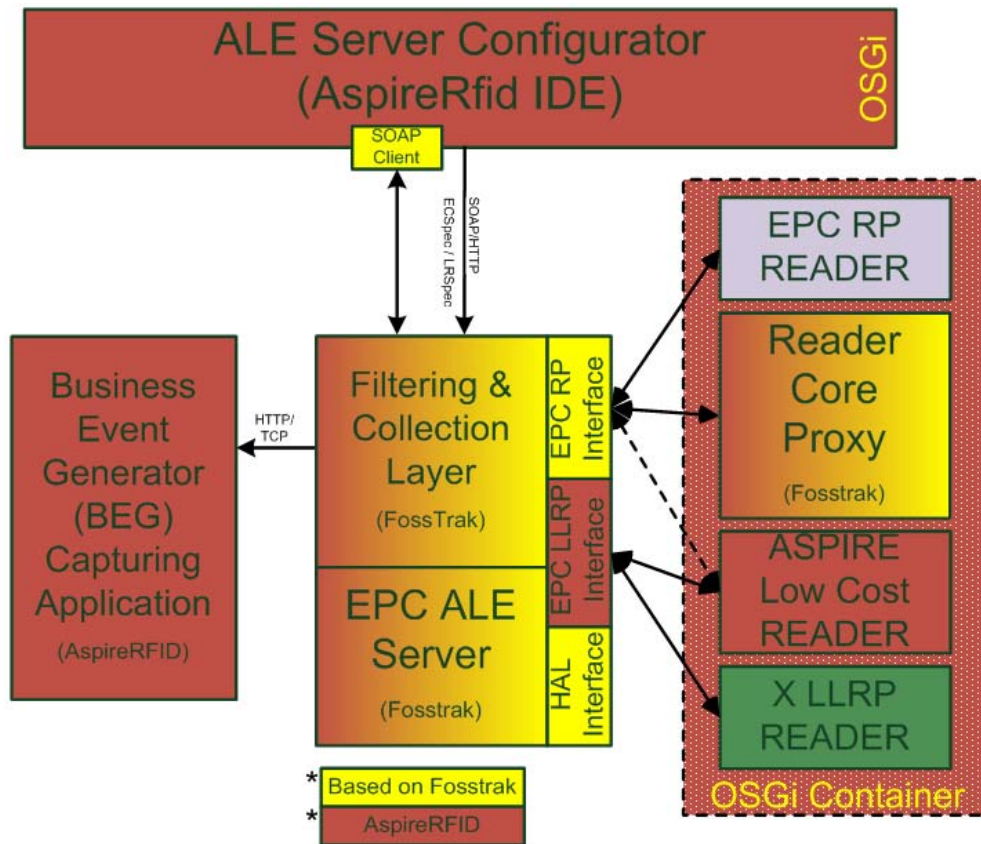

Figure 5 Overview of the AspireRfid Filtering, Collection and ALE solution illustrating components licensed by FossTrak, as well as components entirely developed by AspireRfid

The ALE interface revolves around client requests and the corresponding reports that are produced. Requests can work in:

- *Immediate mode*, in which information is reported on a one-time basis at the time of the request
- Recurring mode, in which information is reported repeatedly whenever an event is detected or at a specified time interval. The results reported in response to a request can be directed back to the requesting client or to a "third party" specified by the requestor.
- *Poll Mode:* The Poll mode of interaction with an ALE server is similar to Immediate, except the ALE client defines the ECSpec on the server ahead of time. It gives the ECSpec a name which allows the ALE client to request data from it using the name instead of sending the ECSpec every time.
- *Subscribe Mode:* Defines the ECSpec ahead of time like Poll. It does not require the ALE client to continually request data from the server. Instead the ALE client adds a subscription to the ECSpec that the ALE server communicates. One example of an ALE subscription would be an ALE client that opens a TCP port that the server can connect to and send reports through it.

The available request modes are shown at the pictures below:
- *Subscribe Mode*: Asynchronous reports from a standing request

Figure 6 Asynchronous reports from a standing request [15]

- ALE XPoll Mode: Synchronous (on-demand) report from a standing request
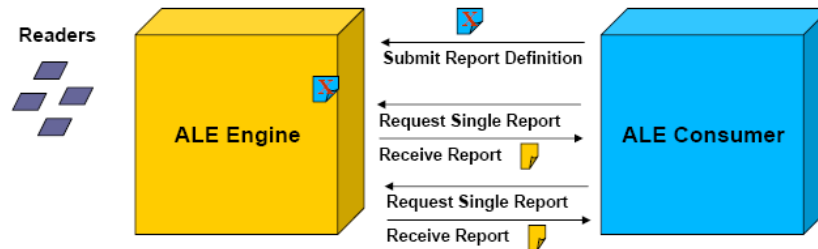

Figure 7 On-demand report from a standing request [15]

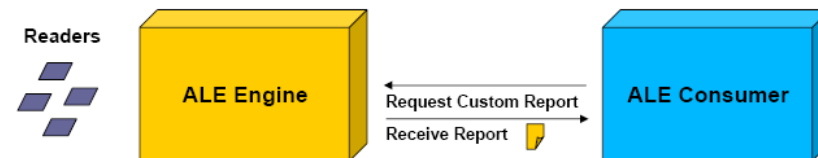- Immediate Mode: Synchronous report from one-time request

Figure 8: Synchronous report from one-time request [15]

3.5 F&C Datatypes

The primary data types associated with the ALE API are:
- Filtering Specifications (e.g., ECSpec according to EPC-ALE [2]), which specify how an event cycle is calculated
- Reports, (e.g., ECReports according to standard [2]), which contains one or more reports generated from a single activation of a filtering specification. Report instances must be provided in both a "pull" and "push" manner. As a result, a related subscription mechanism needs to be implemented.

3.6 F&C Core Engine (Event Cycles)

An event cycle or command cycle is an interval of time over which an ALE Implementation carries out interactions with one or more Readers on behalf of an ALE client.

An event cycle is the smallest unit of interaction between an ALE client and an ALE implementation through the ALE Reading API. An event cycle is an interval of time during which Tags are read in the Reading API. A command cycle is the smallest unit of interaction between an ALE client and an ALE implementation through the ALE Writing API. A command cycle is an interval of time during which Tags are written, or other operations performed upon them, in the Writing API.

3.7 F&C ECSpecs & ECReports

Filtering specifications describe event cycles, along with one or more reports which are to be generated from it. Filtering specifications must typically contain:
- A list of logical readers whose read cycles are to be included in the event cycle.
- A specification of how the boundaries of event cycles are to be determined.
- A list of specifications, each describing a report to be generated from this event cycle.

Note that filtering specifications will generate event cycles as long as there is at least one subscriber to the server.

Reports are the output of an event cycle. Report instances contain a list of reports, each one corresponding to a filtering specification. Moreover, report instances include a number of metadata that provides useful information about the event cycle.

3.8 Logical Readers

Prerequisite to define a filtering specification is the definition of the Logical reader(s). To this end, an application programming interface (API) enabling clients to define logical reader names used with the APIs that access the tags (namely Reading API and Writing API), must be defined. The logical reader API allows also the manipulation of configuration properties associated with logical reader names.

ID: D3.3_Data Collection, Filtering and Application Level
Events.doc10
Revision: 0.10

Date: 29 Apr 2009

Security: Public
Page 19/58

Section 4   ASPIRE Enhancements and Bug Fixes over Licensed Fosstrak modules

AspireRFID for its Filtering and Collection module has licensed and used as its codebase the Open Source project Accada/Fosstrak [1] which was extended and bug fixed to comply to the ASPIRE's Architecture Specifications. The main bug fixes and added features are mentioned below:

- Added EPC LLRP Interface (to support LLRP compliant RFID readers) as shown at Figure 5 above.
- The EPC ALE specifications [2] requires for the Filtering and Collection module to be able to include at its reports the following Tag format types: Tag, RawHex, RawDecimal and EPC. So exept from EPC format that was supported the following changes where made to support the rest of them.
  - Added functions for converting byte[] Tag IDs to raw Decimal, raw HEX and tagURI formats so as the produced ECReport to be able to include these Tag format types.
  - Add check at the Pattern Class method isMember(String tagPureURI, String tagURI) to check whether the requested pattern from the ECSpec is pure id pattern or not so as to use the right Tag format (Pure URI or Tag URI).
  - Add captured tags to the Tag Report Group list as:
    - Raw Decimal
    - Tag URI
    - And Raw Hex
- For now the Filtering and collection module supports only GID-96, SGTIN-64 and SSCC-64 Tag type. For extending the Filtering and Collection server we added SGTIN-96, SGTIN-198, SSCC-96, GSGLN-96, GSGLN-195, GRAI-96, GRAI-170, GIAI-96, GIAI-202, USDOD-96, GID, SGTIN, SSCC, GSGLN, GRAI, GIAI, USDOD to the "PatternType" Object for future support of the rest EPC Tag types **Error! Reference source not found.**.

- Add third field check to the URI pattern (Range is not allowed in pure id patterns as required from the ALE Specifications [2]).
- Clear the faulty read tags (e.g. Invalid URI pattern) in the ECSpecValidationExceptionResponse and the ImplementationExceptionResponse so as Filtering and Collection server implementation wont "stuck" if an exception occur. All previous tags will be deleted so ADDITIONS and DELETIONS will be reset.
- At EventCycle.run() corrected the faulty use of "lastEventCycleTags=tags" which was replacing the hole list of the Tags read with the new list of Tags read and changed into "lastEventCycleTags.addAll(tags)" so as not to lose the previous collected tags before the event cycle finish and the Tags are reported.
- Fixed faulty behavior when using ADDITONS and DELETIONS in the same ECSpec (previously if the same tag group was used in two Report Specs, one for ADDITONS one for DELETIONS, it was adding and deleting tag ids at the ECReports arbitrarily).

- Added the ability to clear the ADDITONS and DELETIONS history at defined amount of event cycles as specified at the EPC ALE 1.1 specifications [2].

Section 5   ALE Extensions

The ALE specification has an extensibility mechanism which allows sending additional information in ALE reports by means of extension elements, as shown in the example of Table 1. Such type of additional information should be stored in the EPCIS. For the purpose of ASPIRE, different types of information will be sent in the ALE reports, such as:

- Sensor readings.
- Raw byte array of information read from the tag's memory
- General purpose information (the URL of a resource, textual info, etc)

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:ECReports        totalMilliseconds="5000"        terminationCondition="DURATION"
specName="ECSpec_additions"   date="2009-03-14T16:24:55.500+02:00"   ALEID="ETHZ-
ALE443245070" xmlns:ns2="urn:epcglobal:ale:xsd:1">
  <reports>
    <report reportName="extendedReport">
      <group>
        <groupList>
          <member>
            <epc>urn:epc:id:gid:145.255.487</epc>
            <tag>urn:epc:tag:gid-96:145.255.487</tag>
           <extension>
            <gpsCoordinates>46.713753475,4.515625</gpsCoordinates>
             <measurementList>
              <measurement>
                <value>1020.0</value>
                <error>0.0</error>
                <unit>Pa</unit>
                <timestamp>1239888391419</timestamp>
                <applicationName>pressure</applicationName>
                <sensorId>pressuresensor</sensorId>
              </measurement>
              <measurement>
                <value>300</value>
                <error>0.0</error>
                <unit>K</unit>
                <timestamp>1239888391419</timestamp>
                <applicationName>temperature</applicationName>
                <sensorId>tempsensor</sensorId>
              </measurement>
             </measurementList>
            </extension>
          </member>
        </groupList>
      </group>
    </report>
  </reports>
</ns2:ECReports>
```

Table 1: ALE extension example

In the case of general purpose information, it has the motivation of letting the information be specific to the applications using ASPIRE middleware. An extensive list of different use cases can illustrate such usage, but only two such use cases are described here. For instance, in the context of the adaptation of NFC devices and applications, one may have survey answers (concerning the object identified by an RFID tag) to be sent along with the tag ID read. Another example is in the context of a warehouse application that can take snapshots when products on a pallet are scanned. The picture files can be sent along as general information in the form of URLs.

Such mechanisms are under implementation in ASPIRE. The ALE extensions for tag memory raw data and for sensor information will be provided as part of ASPIRE middleware, while general purpose information can be implemented as a pluggable mechanism that should be provided by applications that use ASPIRE middleware. By using the ALE's extensibility mechanism provided in applications built on top of ASPIRE can have custom information in their data collection process.

Section 6   ALE Server Configurator tools

As far as ease of development is concerned, the ASPIRE architecture specifies the existence of an IDE, which is conveniently called AspireRFID IDE and which enables the visual management of all configuration files and meta-data that are required for the operation of an RFID solution.
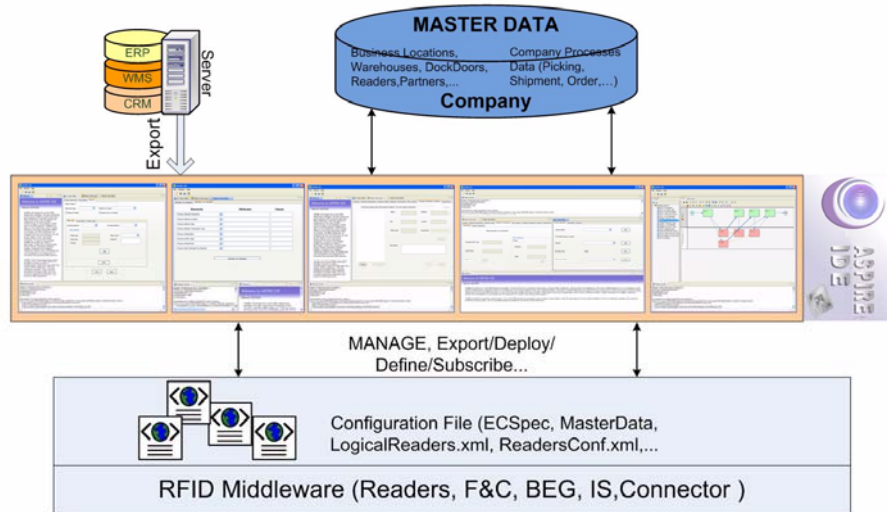

Figure 9 Programmability Tooling

AspireRFID IDE has been designed as an Eclipse RCP (Rich Client Platform) application that runs over Equinox OSGI server. It uses the command API to define menus, pop-up menu items and toolbars so as to support easily plug-ins and provide more control. Every tool is an eclipse plug-in/bundle that is able to be installed or removed as needed. This way many editions of the AspireRFID IDE can be released depending on the functionalities required (as simple or as complicate depending on the demands) for the ASPIRE's RFID middleware blocks that will be used.

The ALE Server Configurator plug-in shown in Figure 10 below is a tool that provides a control client to execute Application Level Event specification (ALE) commands on a component that implements the ALE specification and more specifically the AspireRFID Filtering and Collection Server. The ALE Server Configurator is integrated in the AspireRFID IDE as an Eclipse RCP (Rich Client Platform) plug-in application.

The Implementation and Documentation of ALE Server Configurator can be found at the AspireRFID Forge (http://forge.ow2.org/projects/aspire) and Wiki (http://wiki.aspire.ow2.org/) page respectively. The executable final version of the ALE Server Configurator (AspireRfidIdeToolCollection[version]) can be found at the AspireRFID Forge Files page (http://forge.ow2.org/project/showfiles.php?group_id=324), the source code of the implementation can be found at the AspireRFID SVN(http://forge.ow2.org/plugins/scmsvn/index.php?group_id=324). Directions

on how to use the ALE Server Configurator can be found at the AspireRFID Wiki documentation page http://wiki.aspire.ow2.org/xwiki/bin/view/Main.Documentation.AspireIDE/AleServerConfigurator.

Figure 10 Ale Server Configurator Plug-in

6.1 ECSpec Configurator

ECSpec Configurator is used to manage ECSpecs life cycle by providing a user friendly configurable interface. The ECSpec Configurator implements the ALE main API class using the following methods:

- define(specName : String, spec : ECSpec), which creates a new ECSpec having the name specName, according to spec.
- undefine(specName : String), which removes the ECSpec named specName that was previously created by the define method.
- getECSpec(specName : String), which returns the ECSpec that was provided when the ECSpec named specName was created by the define method.
- getECSpecNames(), which returns an unordered list of the names of all ECSpecs that are visible to the caller.
- subscribe(specName : String, notificationURI : String), which adds a subscriber having the specified notificationURI to the set of current subscribers of the ECSpec named specName.
- unsubscribe(specName : String, notificationURI : String), which removes a subscriber having the specified notificationURI from the set of current subscribers of the ECSpec named specName.
- poll(specName : String), which requests an activation of the ECSpec named specName, returning the results from the next event cycle to complete.

- immediate(spec : ECSpec), which creates an unnamed ECSpec according to spec, and immediately requests its activation.
- getSubscribers(specName : String), which returns an unordered, possibly empty list of the notification URIs corresponding to each of the current subscribers for the ECSpec named specName.
- getStandardVersion(), which returns a string that identifies what version of the specification this implementation of the Reading API complies with.
- getVendorVersion(), which returns a string that identifies what vendor extensions this implementation of the Reading API provides.
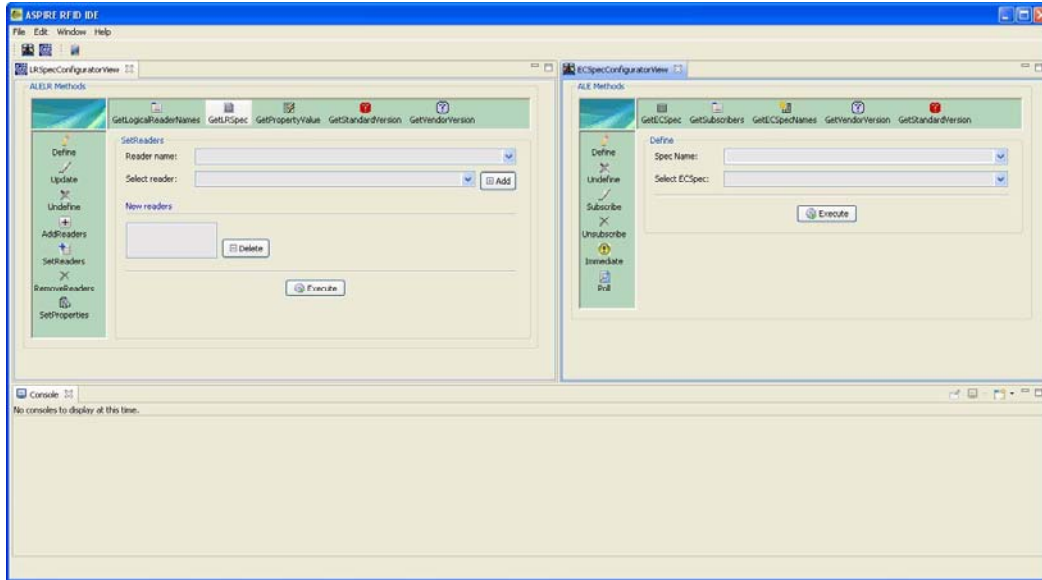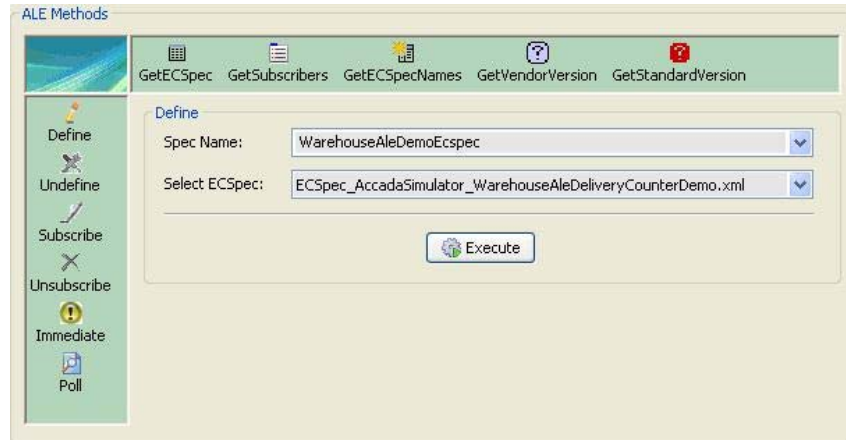

Figure 11 ECSpec Configurator View

6.2   LRSpec Configurator

LRSpec Configurator is used to manage LRSpecs life cycle by providing a user friendly configurable interface. The LRSpec Configurator implements the ALE main Reading API class with the following methods:

- define(name : String, spec : LRSpec), which creates a new logical reader named name according to spec.
- update(name : String, spec : LRSpec) , which changes the definition of the logical reader named name to match the specification in the spec parameter.
- undefine(name : String) , which removes the logical reader named name.
- getLogicalReaderNames(), which returns an unordered list of the names of all logical readers that are visible to the caller.
- getLRSpec(name : String) , which returns an LRSpec that describes the logical reader named name.
- addReaders(name : String, readers : List<String>), which adds the specified logical readers to the list of component readers for the composite logical reader named name.
- setReaders(name : String, readers : List<String>), which changes the list of component readers for the composite logical reader named name to the specified list.

ID: D3.3_Data Collection, Filtering and Application Level
Events.doc10
Revision: 0.10
Date: 29 Apr 2009

Security: Public
Page 26/58

- removeReaders(name : String, readers : List<String>), which removes the specified logical readers from the list of component readers for the composite logical reader named name.
- setProperties(name : String, properties : List<LRProperty>), which changes properties for the logical reader named name to the specified list.
- getPropertyValue(name : String, propertyName : String) , which returns the current value of the specified property for the specified reader, or null if the specified reader does not have a property with the specified name.
- getStandardVersion(), which Returns a string that identifies what version of the specification this implementation of the ALE Logical Reader API complies with.
- getVendorVersion(), which returns a string that identifies what vendor extensions of the ALE Logical Reader API this implementation provides.

LRSpec configurator currently supports RP (Reader Protocol), LLRP (Low Level Reader Protocol), generic HAL (Hardware Abstraction Layer) and Composite readers.
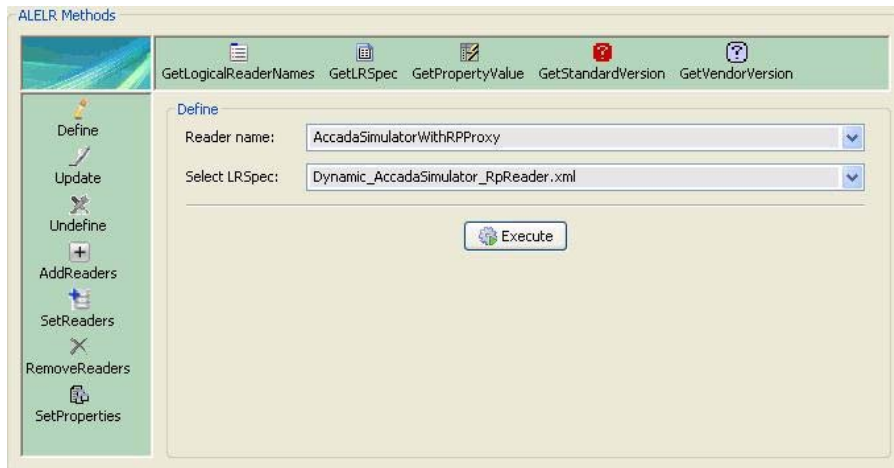

Figure 12 LRSpec Configurator View

Section 7   F&C Practical Use Case Examples and Implementation

In this Section, three simple use cases that demonstrate the additions, deletions and grouping capabilities of the Filtering and Collection server will be presented corresponding to a Company's needs to track products added, removed or counted from a shelf.

7.1   Use case 1: Specific family's products added to a shelf

This use case's purpose is to identify the set of Tag ID's of the Class "*urn:epc:pat:gid-96:145.255.**" that have been added to a specific shelf. For capturing and viewing the produced ECReport from the Filtering and Collection server a rather simple application will be used called "TCP Message Capturer".

The corresponding ECSpec that is defined and subscribed to the Filtering and Collection Server to serve the needs of this use case is shown in the Table 2 below.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:ECSpec includeSpecInReports="true" xmlns:ns2="urn:epcglobal:ale:xsd:1">
  <logicalReaders>
    <logicalReader>AccadaSimulatorWithRPProxy</logicalReader>
  </logicalReaders>
  <boundarySpec>
    <repeatPeriod unit="MS">5000</repeatPeriod>
    <duration unit="MS">5000</duration>
    <stableSetInterval unit="MS">0</stableSetInterval>
    <extension/>
  </boundarySpec>
  <reportSpecs>
    <reportSpec reportOnlyOnChange="false" reportName="additionsReport"
     reportIfEmpty="true">
      <reportSet set="ADDITIONS"/>
      <filterSpec>
        <includePatterns>
          <includePattern>urn:epc:pat:gid-96:145.255.*</includePattern>
        </includePatterns>
      </filterSpec>
      <output includeTag="true" includeRawHex="false" includeRawDecimal="false"
       includeEPC="true" includeCount="false"/>
    </reportSpec>
  </reportSpecs>
  <extension/>
</ns2:ECSpec>
```

Table 2: Use case 1 ECSpec

The ECSpec is defined by using the ECSpecConfigurator tool. Afterwards, the ECSpec is subscribed, by using the ECSpecConfigurator tool, to the ASPIRE TCP Message Capturer. A tag that belongs to the "*urn:epc:pat:gid-96:145.255.**"

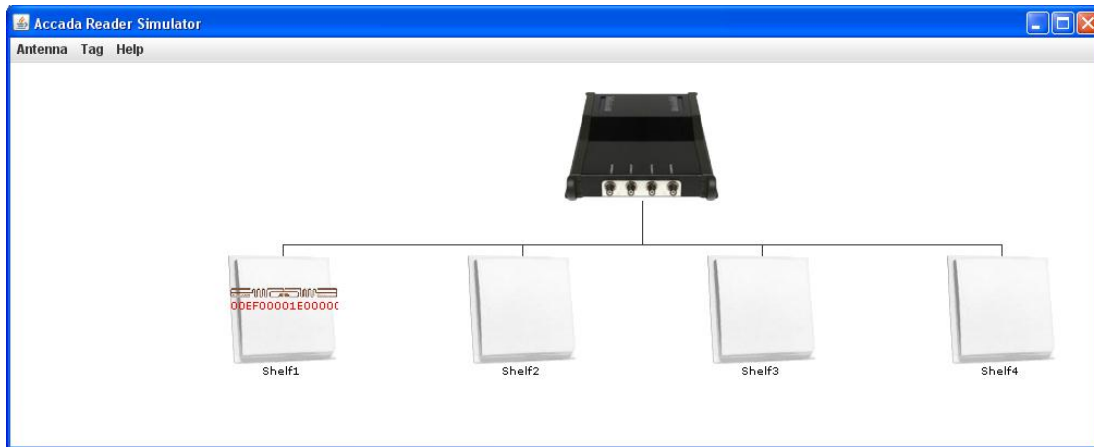pattern is placed within the proximity of the shelf antenna as shown in picture Figure 13 below.

Figure 13 Fosstrak Reader Simulator.

The produced report from the F&C server is captured by the ASPIRE TCP Message Capturer and is shown in Table 3 below.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:ECReports        totalMilliseconds="5000"        terminationCondition="DURATION"
specName="ECSpec_additions"   date="2009-03-14T16:24:55.500+02:00"   ALEID="ETHZ-
ALE443245070" xmlns:ns2="urn:epcglobal:ale:xsd:1">
  <reports>
    <report reportName="additionsReport">
      <group>
        <groupList>
          <member>
            <epc>urn:epc:id:gid:145.255.487</epc>
            <tag>urn:epc:tag:gid-96:145.255.487</tag>
          </member>
        </groupList>
      </group>
    </report>
  </reports>
</ns2:ECReports>


-----Report 2------

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:ECReports        totalMilliseconds="5000"        terminationCondition="DURATION"
specName="ECSpec_additions"   date="2009-03-14T16:24:55.500+02:00"   ALEID="ETHZ-
ALE443245070" xmlns:ns2="urn:epcglobal:ale:xsd:1">
  <reports>
    <report reportName="additionsReport">
      <group>
        <groupList>
          <member>
          </member>
        </groupList>
      </group>
    </report>
  </reports>
</ns2:ECReports>
```

Table 3: Use case 1 ECReport

As we can see from the produced ECReport the tag id is reported only once as soon as the product enters the shelf reader proximity and never again.

7.2   Use case 2: Specific family's products removed from a shelf

This use case's purpose is to identify the set of Tag ID's of the following Class "*urn:epc:pat:gid-96:145.255.**"* that have been removed from a specific shelf.

The corresponding ECSpec that is defined and subscribed to the Filtering and Collection Server to serve the needs of this use case is shown in Table 4 below.

ID: D3.3_Data Collection, Filtering and Application Level
Events.doc10
Revision: 0.10

Date: 29 Apr 2009

Security: Public
Page 30/58

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:ECSpec includeSpecInReports="true" xmlns:ns2="urn:epcglobal:ale:xsd:1">
  <logicalReaders>
    <logicalReader>AccadaSimulatorWithRPProxy</logicalReader>
  </logicalReaders>
  <boundarySpec>
    <repeatPeriod unit="MS">5000</repeatPeriod>
    <duration unit="MS">5000</duration>
    <stableSetInterval unit="MS">0</stableSetInterval>
    <extension/>
  </boundarySpec>
  <reportSpecs>
    <reportSpec reportOnlyOnChange="false" reportName="deletionsReport"
     reportIfEmpty="true">
      <reportSet set="DELETIONS"/>
      <filterSpec>
        <includePatterns>
          <includePattern>urn:epc:pat:gid-96:145.255.*</includePattern>
        </includePatterns>
      </filterSpec>
      <output includeTag="true" includeRawHex="true" includeRawDecimal="false"
       includeEPC="true" includeCount="false"/>
    </reportSpec>
  </reportSpecs>
  <extension/>
</ns2:ECSpec>
```

Table 4: Use case 2 ECSpec

The ECSpec is defined by using the ECSpecConfigurator tool. Afterwards, the ECSpec is subscribed, by using the ECSpecConfigurator tool, to the ASPIRE TCP Message Capturer. A tag that belongs to the *"urn:epc:pat:gid-96:145.255.*"* pattern that is already placed within the proximity of the shelf antenna is removed.

The produced report from the F&C server is captured by the ASPIRE TCP Message Capturer and is shown in Table 5 below.

ID: D3.3_Data Collection, Filtering and Application Level
Events.doc10
Revision: 0.10

Date: 29 Apr 2009

Security: Public
Page 31/58

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:ECReports      totalMilliseconds="5000"      terminationCondition="DURATION"
specName="ECSpec_deletions"  date="2009-03-14T16:34:32.359+02:00"  ALEID="ETHZ-
ALE443245070" xmlns:ns2="urn:epcglobal:ale:xsd:1">
   <reports>
     <report reportName="deletionsReport">
       <group>
         <groupList>
           <member>
              <epc>urn:epc:id:gid:145.255.487</epc>
              <tag>urn:epc:tag:gid-96:145.255.487</tag>
              <rawHex>urn:epc:raw:96.x3500000910000FF0000001E7</rawHex>
           </member>
         </groupList>
       </group>
     </report>
   </reports>
</ns2:ECReports>


----Report 2----

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:ECReports      totalMilliseconds="5000"      terminationCondition="DURATION"
specName="ECSpec_deletions"  date="2009-03-14T16:34:32.359+02:00"  ALEID="ETHZ-
ALE443245070" xmlns:ns2="urn:epcglobal:ale:xsd:1">
   <reports>
     <report reportName="deletionsReport">
       <group>
         <groupList>
           <member>
           </member>
         </groupList>
       </group>
     </report>
   </reports>
</ns2:ECReports>
```

Table 5: Use case 2 ECReport

As we can see from the produced ECReport the tag id is reported only once as soon as the product is removed from the shelf reader proximity and never again.

7.3 Use case 3: Specific family's products' grouping and counting

This use case's purpose is to identify, group and count the set of Tag ID's of the following Tag id Class "*urn:epc:pat:gid-96:145.255.**" that have been passed thru an RFID dock door.

The corresponding ECSpec that is defined and subscribed to the Filtering and Collection Server to serve the needs of this use case is shown in Table 6 below.

ID: D3.3_Data Collection, Filtering and Application Level
Events.doc10
Revision: 0.10

Date: 29 Apr 2009

Security: Public
Page 32/58

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:ECSpec includeSpecInReports="true" xmlns:ns2="urn:epcglobal:ale:xsd:1">
  <logicalReaders>
    <logicalReader>AccadaSimulatorWithRPProxy</logicalReader>
  </logicalReaders>
  <boundarySpec>
    <repeatPeriod unit="MS">4500</repeatPeriod>
    <duration unit="MS">4500</duration>
    <stableSetInterval unit="MS">0</stableSetInterval>
    <extension/>
  </boundarySpec>
  <reportSpecs>
    <reportSpec reportOnlyOnChange="false"
     reportName="currentTagsGroupingReport" reportIfEmpty="true">
      <reportSet set="CURRENT"/>
      <groupSpec>
        <pattern>urn:epc:pat:gid-96:145.255.*</pattern>
      </groupSpec>
      <output includeTag="true" includeRawHex="false" includeRawDecimal="false"
       includeEPC="true" includeCount="true"/>
    </reportSpec>
  </reportSpecs>
  <extension/>
</ns2:ECSpec>
```

Table 6: Use case 1 ECSpec

The ECSpec is defined by using the ECSpecConfigurator tool. Afterwards, the ECSpec is subscribed, by using the ECSpecConfigurator tool, to the ASPIRE TCP Message Capturer. Three tags that belongs to the "*urn:epc:pat:gid-96:145.255.*"* pattern pass thru the RFID dock door.

The produced report from the F&C server is captured by the ASPIRE TCP Message Capturer and is shown in Table below.

ID: D3.3_Data Collection, Filtering and Application Level
Events.doc10
Revision: 0.10

Date: 29 Apr 2009

Security: Public
Page 33/58

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:ECReports      totalMilliseconds="4500"      terminationCondition="DURATION"
specName="ECSpec_current"    date="2009-03-14T16:51:47.078+02:00"    ALEID="ETHZ-
ALE443245070" xmlns:ns2="urn:epcglobal:ale:xsd:1">
  <reports>
    <report reportName="currentTagsGroupingReport">
      <group groupName="urn:epc:pat:gid-96:145.255.*">
        <groupList>
          <member>
            <epc>urn:epc:id:gid:145.255.487</epc>
            <tag>urn:epc:tag:gid-96:145.255.487</tag>
          </member>
          <member>
            <epc>urn:epc:id:gid:145.255.485</epc>
            <tag>urn:epc:tag:gid-96:145.255.485</tag>
          </member>
          <member>
            <epc>urn:epc:id:gid:145.255.489</epc>
            <tag>urn:epc:tag:gid-96:145.255.489</tag>
          </member>
        </groupList>
        <groupCount>
          <count>3</count>
        </groupCount>
      </group>
    </report>
  </reports>
</ns2:ECReports>
```

Table 7: Use case 3 ECReport

As we can see from the produced ECReport the Tag ids are grouped under the group name they belong to (urn:epc:pat:gid-96:145.255.*) and the tag count is included also.

ID: D3.3_Data Collection, Filtering and Application Level
Events.doc10
Revision: 0.10

Date: 29 Apr 2009

Security: Public
Page 34/58

Section 8   Aspire Middleware modularity and dynamic provisioning using OSGi

8.1   The OSGi dynamic service platform

The OSGi framework (http://www.osgi.org) is a dynamic service platform for the construction of modular Java applications, allowing the installation, uninstallation, update and startup of components without needing to do an application reboot. In OSGi, components can provide services that are registered in a service registry which is part of the OSGi framework. Other components can then consume such services in a loose coupled way without knowing the actual implementation behind the service interface, a sort of Service Oriented Architecture (SOA) but with objects (services) inside the same JVM. However, special care must be taken concerning the dynamicity of components and services that may both appear and disappear during application execution. Due to the high dynamicity characterizing OSGi components and services, availability of the latter is likely to evolve at runtime. This phenomenon results in intermittent services provided in a discontinuous way. Application developers should face the issue of handling service disruptions in their code, which is being addressed as part of the UJF work [19], as well as the correct releasing of service instances [20].

The Aspire middleware is partially modularized as OSGi bundles (ie artifacts). Those module can export or import packages (ie Java libraries) or/and exchanging services that are dynamically registered (and unregistered).

8.2   The OSGi dynamic provisioning

The OSGi Bundle Repository (OBR RFC 112 http://www.osgi.org/Download/File?url=/download/rfc-0112_BundleRepository.pdf) is a concept of a repository that stores information (e.g. download URL, version, dependencies on other components) about a set of components (bundles). There are OBR tools, in the form of components themselves, which can be deployed in any OSGi framework. Such tools are able to parse the contents of an OBR (identified by a URL, either local or remote) and provide functionalities such as listing and installing components located in an OBR. By using the OBR to install a component, the dependencies are automatically installed along with the component, thus minimizing deployment efforts and time.

The Aspire middleware uses facilities such as the OBR service to deploy (ie installed, updated) a set of bundles and their explicit dependencies. More refined services such as the OW2 JOnAS deployer service are envisaged. This advanced deployer will cover features such as the update of mission-critical applications such as RFID applications in non-stop production environment.

8.3   The OSGi WireAdmin services

The OSGi specification proposes facilities to manage connections between sensors data producers and consumers through its WireAdmin service using a SOA approach. Producers and consumers are modeled as uniquely identified OSGi services (i.e. they are published in a service registry along with a set of properties). They are delivered in deployment units called bundles. At runtime the connectors, namely wires, are managed by the WireAdmin Service. This service allows wires to be created, deleted, retrieved and updated programmatically. Once connected, producers can either push data into consumers or provide data when they are polled through the wires. Wires are persistent entities that bind specific producers and consumers through unique identifiers. In order to handle the dynamicity of devices such as readers or sensors, AspireRFID uses W-ADL and WireAdminBinder [21]. Collection of wires are described in a declarative language called Wired Application Description Language (WADL), so that the WireAdminBinder interpreter can create and destroy the connectors between measurement producers and consumers as they are introduced or removed dynamically from the execution environment.

The Aspire middleware relies on this producer-consumer model to represent sensors and collected the data in order to add them in the ECReport extensions. Sensors drivers are automatically wired to the F&C using the WireAdminBinder.

8.4   Service Oriented Component Models for the OSGi platform

On the OSGi platform, several component models have aimed at handling dynamism concerns in order to facilitate the development of dynamic service-based applications. The first was ServiceBinder developed by the UJF Adele team. ServiceBinder became later Declarative Services in the 4th release of the specification. Spring-Dynamic Modules is another component model that uses XML descriptions to automate bindings in the Spring framework. The iPOJO component model (http://felix.apache.org/site/apache-felix-ipojo.html originally developed by the UJF Adele team) follows the same principles but it targets a larger spectrum of applications and hosts from embedded systems (ARM9) to high-end servers (JavaEE clusters). Components declare their service dependencies and according to these declarations, component bindings are automated dynamically at runtime, although it is still possible to declare static mandatory services that cannot be substituted. Regarding iPOJO, the framework extensibility allows the container to manage transparently other non-functional via a handler mechanism such as persistence using Hibernate or distribution using UPnP or Web Services (Apache XFire/CXF).

The Aspire middleware is partially componentized as iPOJO component factories and instances (mainly code in the UJF branch). The current components cover several readers, several sensors and several actuators.
The main iPOJO handler that are currently used are
*   the EventAdminHandler (http://felix.apache.org/site/event-admin-handlers.html) for event publication and subscription

- the JMX Handler (http://felix.apache.org/site/ipojo-jmx-handler.html) to expose administration functionalities according the RFC 0139.

More OSGi services will be progressively "iPOJOfied". More iPOJO handlers will be progressively used: CM, DSLA, Distribution (RFC 119 http://www.osgi.org/download/osgi-4.2-early-draft3.pdf).

Section 9   Lower Level Filtering Investigations

So far, we have described every filtering, aggregation and collection that can be operated on raw data before being forwarded to upper applications. These functionalities are, as mentioned, part of the ALE standard. As explained above, the functionalities are developed and offered in the core of the middleware.

Nevertheless, in order to reduce the network load and to facilitate operations off-line as specified in the ASPIRE objectives, it would be interesting to deport part of the ALE engine to the reader or mobile terminal. Indeed, if filtering and aggregation is allowed at the very lowest level, the network is less loaded and will scale better. Note that this topic is also part of Task 3.2. Therefore, a detail report on the implementation of low level filtering will be presented in D3.4b due M29. Whereas the current document will document investigations and preliminary work conducted on low level filtering to date.

To make the needs for low level filtering clearer, let's take an example to illustrate it. Let us suppose that a big store needs to draw up an inventory. The employee in charge of the inventory has at his disposal a mobile reader with a terminal on which he can view objects in real time read by the reader. This reader is wireless and may be disconnected from the network. Such an application may be performed in "real time", meaning that once a tag is read, its EPC is sent immediately to the middleware core to be filtered and aggregated.
If the reader is able to perform these filtering and aggregating tasks, less data are sent through the network. Nevertheless, all these actions cannot be embedded on the mobile terminal, according to the specific rules and the CPU and memory capacities of the terminal. Therefore, we propose to tune the filtering rules according to the specific applications and embed only the ones needed. The easiest and simplest form of low level filtering is to delete duplicated data. For more complex actions, embedding specific filtering rules on the terminal supposes that before starting the inventory (or any other action), the terminal needs to be connected to the middleware and to exchange this information.

We are currently developing this module which deports adaptive filtering rules to the terminal. This is also part of the WP5 "Interface with the reader".
This module has two roles. First, it has to create and transfer filtering rules depending of the application needs and second, it has to forward the aggregated and filtered data to the middleware, which has to process them together with data coming from other readers that are not necessarily aggregated.

Following the SME needs, we found out that such inventories are generally performed shelf by shelf, by kind of objects. When a shelf is scanned, some objects may be read at the same time since entering the range of the reader while lying on a neighboring shelf. A low-level filtering should allow deleting these parasite data.

ID: D3.3_Data Collection, Filtering and Application Level
Events.doc10
Revision: 0.10

Date: 29 Apr 2009

Security: Public
Page 38/58

To do so, we first investigated a solution based on the Received Signal Strength indicator (RSSI). The idea was the following. Object tags lying on the shelf currently scanned are closer to the reader than objects tags lying on further shelves. By measuring the RSSI, we should remove all tags with a low RSSI. Though seeming promising and interesting, first experiment results rose technical problems that makes this RSSI tracks not feasible within the time and with the resources dedicated to the ASPIRE project. In fact, experimentations showed the impossibility to simply map a RSSI to a distance in a proportional fashion. A tag close to the reader can induce a RSSI smaller than a further tag. This is due to the fact that the power they can induce from the electromagnetic field depends on their orientation. And as tags do not always have the same orientation, the RSSI is not directly proportional to the distance. Elaborating an accurate mapping between the RSSI and the distance would take a longer time and need further experiments. In addition, we believe that this relationship is highly hardware-dependent. Since ASPIRE is needed to cover a scope as large as possible, we gave up with this track, at least temporarily.

The second idea we investigated is the following one. If the terminal is able to embed a list of identifiers, before starting the inventory, the reader has to download the list of potential tags lying on neighboring shelves. From this list, the reader is able to filter the parasite data. This obviously requires a higher amount of available memory but allows a lower level filtering and thus reduces the amount of data sent to the middleware. By doing so, the reader will leverage data of objects lying on the currently scanned shelves. This allows two things. First, it draws up an accurate inventory of a kind of articles and second, it locates objects placed on the wrong shelf since the identifier of that object will be leveraged within the set of data. The only anomaly that will not be notified happens when an object belonging to a neighboring shelf is on the scanned shelf. Nevertheless, SME claim that this is an error they can deal with.

Another challenge that arises is pertaining to the maintenance of the quality of data collected. One of ASPIRE's properties states that ASPIRE will maintain data quality which is one of the recommendations of ePrivacy and other data directives. Data Quality means ASPIRE middleware based applications would not collect or process information that is not essential to the organisation. For instance, if a person with a tagged watch walks in to a store which implements an ASPIRE middleware system; the readers will not collect the data by filtering out the detection of the watch.

By defining filters or implementing low level filtering using the second method illustrated above where parasite data are deleted, organisations would not be able to detect that the person who had walked into the store is wearing a certain watch. However, the challenge that arises here is how would the ASPIRE middleware prevent adopters to configure the ECSpecs to read all tagged objects in the store even though those objects do not belong to the store. And if they do, what system could be set so that flags or alert messages could be used to warn the implementers and notify auditors to maintain the Data Quality and comply

ID: D3.3_Data Collection, Filtering and Application Level
Events.doc10
Revision: 0.10

Date: 29 Apr 2009

Security: Public
Page 39/58

with various privacy directives. These are issues that need to be tackled in later tasks and would be defined more precisely in the next deliverables.

To conclude this section, we detailed here the first low level investigations and experiments. We intend to implement the abovementioned track and to analyse it. We need to compare it to a solution that does not apply any low level filtering and send the raw data to the middleware. We need to quantify what amount of data is saved against what amount of memory is needed in order to specify when one solution should be chosen instead of another.

Section 10   Conclusions

This deliverable has provided a detailed description of data collection, filtering and application level event functionalities of the ASPIRE RFID middleware platform. The document was particularly concentrated on the specifications of the interfaces, the modes of operation, the messages supported and the rules of interaction of the filtering and collection server with other modules of the ASPIRE architecture. Emphasis has been put on illustrating the genuine features of the AspireRfid F&C middleware server, while the general operation of the ALE server (as part of the FossTrak licensed server is illustrated in an Appendix).

The document also presented specific examples of the filtering and collection functionalities and some initial research efforts towards the implementation of low level filtering functionalities to be hosted at the reader or interrogators. Future research issues include the investigation of reader interference management, which is an increasingly complex problem in current mobile RFID deployments, and the optimization of the anti-collision or singulation mechanisms from the perspective of the middleware platform.

The deliverable has also presented the implementation status of the different functionalities of the ALE server and its APIs, thus serving as a guide for ASPIRE developers. The contents of this deliverable are complemented with an accompanying CD which contains the actual software developed for the data collection, filtering and application level events functionalities.

Section 11   List of Figures

ID: D3.3_Data Collection, Filtering and Application Level
Events.doc10
Revision: 0.10

Date: 29 Apr 2009

Security: Public
Page 42/58

Section 12   List of Tables

Section 13   List of Acronyms

ALE	Application Level Event
API	Application Product Interface
ASPIRE	Advanced Sensors and lightweight Programmable middleware for Innovative Rfid Enterprise applications
BEG	Business Event Generator
BSS	Base Service Set
CSS	Configuration Service Set
EPC	Electronic Product Code
EPCIS	Electronic Product Code Information Services
F&C	Filtering and Collection
HAL	Hardware Abstraction Layer
HTTP	HiperText Transfer Protocol
IDE	Integrated Development Environment
iPOJO	injected POJO
JMX	Java Management Extensions
LLRP	Low Level Reader Protocol
MSS	Monitoring Service Set
OBR	OSGi Bundle Repository
OSGI	Open Service Gateway Initiative
OSI	Open System Interconnection
OSS	Open Source Software
POJO	Plain Old Java Object
RFID	Radio Frequency Identification
RP	Reader Protocol
SME	Small and Medium Enterprise
SNMP	Simple Network Management Protocol
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
TCO	Total Cost of Ownership
TCP	Transfer Control Protocol
UML	Universal Markup Language
WADL	Wired Application Description Language
WP	Work Package
XML	Extensible Markup Language

Section 14   References and bibliography

[1] FossTrak Project, http://www.fosstrak.org/index.html
[2] EPCglobal, "The Application Level Events (ALE) Specification, Version 1.1", February. 2008, available at: http://www.epcglobalinc.org/standards/ale
[3] EPCglobal, "Low Level Reader Protocol (LLRP), Version 1.0.1, August 13", 2007, available at: http://www.epcglobalinc.org/standards/llrp
[4] EPCglobal, "Reader Protocol Standard, Version 1.1, June 21", 2006 available at: http://www.epcglobalinc.org/standards/rp
[5] EPCglobal, "EPCglobal Tag Data Standards, Version 1.4", June 11, 2008, available at: http://www.epcglobalinc.org/standards/tds/
[6] EPCglobal, "EPCglobal Tag Data Translation (TDT) 1.0", January 21, 2006 available at: http://www.epcglobalinc.org/standards/tdt/
[7] LLRP Toolkit, http://www.llrp.org/
[8] Matthias Lampe, Christian Floerkemeier, "High-Level System Support for Automatic-Identification Applications", In: Wolfgang Maass, Detlef Schoder, Florian Stahl, Kai Fischbach (Eds.): Proceedings of Workshop on Design of Smart Products, pp. 55-64, Furtwangen, Germany, March 2007.
[9] Christian Floerkemeier, Christof Roduner, and Matthias Lampe, 'RFID Application Development with the Accada Middleware Platform', IEEE Systems Journal, Vol. 1, Issue 2, pp.82-94, December 2007.
[10]   C. Floerkemeier and S. Sarma, "An Overview of RFID System Interfaces and Reader Protocols", 2008 IEEE International Conference on RFID, The Venetian, Las Vegas, Nevada, USA, April 16-17, 2008.
[11]   Russell Scherwin and Jake Freivald, Reusable Adapters: The Foundation of Service-Oriented Architecture, 2005.
[12]   Java Management Extensions (JMX) Technology Overview, available at: http://java.sun.com/j2se/1.5.0/docs/guide/jmx/overview/architecture.html
[13]   Architecture Review Committee, "The EPCglobal Architecture Framework," EPCglobal, July 2005, available at: http://www.epcglobalinc.org.
[14]   Achilleas Anagnostopoulos, John Soldatos and Sotiris G. Michalakos, 'REFiLL: A Lightweight Programmable Middleware Platform for Cost Effective RFID Application Development', Journal of Pervasive and Mobile Computing (Elsevier), Vol. 5, Issue 1, February 2009, pp. 49-63.
[15]   Application Level Events 1.1(ALE 1.1) Overview, Filtering & Collection WG, EPCglobal,   March   5,   2008   ,   available   at: http://www.epcglobalinc.org/standards/ale
[16]   EPCglobal Inc™. Frequently Asked Questions - ALE 1.1. EPCglobal. [Online] http://www.epcglobalinc.org/standards/ale.
[17]   John Soldatos, "The AspireRfid Project: Is Open Source RFID Middleware still an option?", RFID World, March 16th, 2009.
[18]   K. S. Leong, M. L. Ng, and P. H. Cole, "The Reader Collision Problem in RFID Systems," in Proceedings of IEEE 2005 International Symposium on Microwave, Antenna, Propagation and EMC Technologies for Wireless Communications (MAPE 2005), Beijing, China, 2005.

[19] Lionel Touseau, Walter Rudametkin and Didier Donsez, Towards a SLA-based Approach to Handle Service Disruptions", IEEE International Conference on Services Computing (SCC'08)

[20] Kiev Gama, Didier Donsez, "Service Coroner: A Diagnostic Tool for Locating OSGi Stale References," seaa, pp.108-115, 2008 34th Euromicro Conference Software Engineering and Advanced Applications, 2008

[21] Lionel Touseau, Humberto Cervantes and Didier Donsez, "An Architecture Description Language for Dynamic Sensor-Based Applications", 5th International IEEE Consumer Communications & Networking Conference (CCNC'08)

APPENDIX I – ALE Server Operation (EPC-ALE) and Licensed Accada/FossTrak Implementation

ALE Built-in Datatypes and Formats (Accada/Fosstrak Implementation)

This section defines data types and formats that are supported by the ALE Server as described in [2]. The ALE Server recognizes each data type and format defined in this section and interprets it as defined herein.

In general, the specification of each data type has to say what formats may be used with that data type. A format must define syntax for literal values, for filter patterns, and for grouping patterns.

- The epc datatype
  The epc datatype refers to the space of values defined in the EPCglobal Tag Data Standard [TDS1.3.1]. Because this includes "raw" EPC values, any bit string of any length may be considered a member of the epc datatype.
    - o Binary Encoding and Decoding of the EPC Datatype
    - o EPC datatype Formats
    - o EPC datatype Pattern Syntax
    - o EPC datatype Grouping Pattern Syntax

- Unsigned Integer (uint) Datatype
  ALE Server recognizes the string uint as a valid datatype as specified in this section.
  The space of values for the datatype uint is the set of non-negative integers.
    - o Binary Encoding and Decoding of the Unsigned Integer Datatype
    - o Unsigned Integer Datatype Formats
    - o Unsigned Integer Pattern Syntax
    - o Unsigned Integer Grouping Pattern Syntax

- The bits Datatype
  The space of values for the datatype bits is the set of all non-empty and finite-length sequences of bits.
    - o Binary Encoding and Decoding of the Bits Datatype
    - o Bits Datatype Formats

ALE Reading API Implementation (Licensed Accada/Fosstrak Implementation)

This interface makes use of a number of complex data types. The most significant are the ECSpec, which specify how an event cycle is to be calculated, and the ECReports, which contains one or more reports generated from one activation of an ECSpec. Through the ALE interface clients may define and manage event cycle specifications (ECSpecs), read Tags on-demand by activating ECSpecs synchronously, and enter standing requests (subscriptions) for ECSpecs to be activated asynchronously. Results from standing requests are delivered through the ALECallback interface.

ASPIRE expose the ALE interface of the ALE Reading API via a wired protocol, more specifically TCP-IP, and via a direct API in which clients call directly into code that implements the API. Likewise, ASPIRE implements the ALECallback interface via HTTP/TCP in which clients receive asynchronous results through a direct callback. The ALE Reading API is described in details in the Filtering and Collection Core specifications [2]. In this section we will have an overview of it.

Reading API Data Types

ECSpec

An ECSpec describes an event cycle and one or more reports that are to be generated from it. It contains:
- a list of logical Readers whose data are to be included in the event cycle,
- a specification of how the boundaries of event cycles are to be determined,
- a list of specifications, each describing a report to be generated from this event cycle.

The ALE Server interprets the fields of an ECSpec as follows:
- ECLogicalReaders
  It is an unordered list that specifies one or more logical readers that are used to acquire tags.
- ECBoundarySpec
  It specifies how the beginning and end of event cycles are to be determined.
- ECTime
  It denotes a span of time measured in physical time units.
- ECTimeUnit
  It is an enumerated type denoting different units of physical time that may be used in an ECBoundarySpec.
- ECTrigger
  It denotes a URI that is used to specify a start or stop trigger for an event cycle or command cycle.
- ECReportSpec
  It specifies one report to be included in the list of reports that results from executing an event cycle. An ECSpec contains a list of one or more

ECReportSpec instances. When an event cycle completes, an ECReports instance is generated, unless suppressed. An ECReports instance contains one or more ECReport instances, each corresponding to an ECReportSpec instance in the ECSpec that governed the event cycle. The ECReports contained inside an ECReport cannot contain any ECReports. The number of ECReport instances may be fewer than the number of ECReportSpec instances, due to the rules of suppression of individual ECReport instances.

- ECReportSetSpec
  It is an enumerated type denoting what set of Tags is to be considered for filtering and output: all Tags read in the current event cycle, additions from the previous event cycle, or deletions from the previous event cycle.
- ECFilterSpec
  It specifies which Tags are to be included in the final report.
- ECFilterListMember
  It specifies filtering by comparing a single field of a Tag to a set of patterns. This type is used in both the Reading API and the Writing API.
- ECGroupSpec
  It defines how filtered EPCs are grouped together for reporting.
- ECReportOutputSpec
  It specifies how the final set of EPCs is to be reported.
- ECReportOutputFieldSpec
  It specifies a Tag field to be included in an event cycle report.
- ECFieldSpec
  It encodes a fieldspec which specifies three things:
    o The fieldname
    o The datatype
    o And the format
- ECStatProfileName
  Each valid value of ECStatProfileName names a statistics profile that can be included in an ECReports.

ECReports

ECReports is the output from an event cycle. The "meat" of an ECReports instance is the ordered list of ECReport instances, each corresponding to an ECReportSpec instance in the event cycle's ECSpec, and appearing in the order corresponding to the ECSpec. In addition to the reports themselves, ECReports contains a number of "header" fields that provide useful information about the event cycle. The ALE Server includes these fields according to the following definitions:

- ECInitiationCondition
  Indicates what kind of event caused the event cycle to initiate: the receipt of an explicit start trigger, the expiration of the repeat period, or a transition to the requested state when no start triggers were specified in the ECSpec. These correspond to the possible ways of specifying the start of an event cycle
- ECTerminationCondition
  Indicates what kind of event caused the event cycle to terminate: the receipt of an explicit stop trigger, the expiration of the event cycle

ID: D3.3_Data Collection, Filtering and Application Level
Events.doc10
Revision: 0.10

Date: 29 Apr 2009

Security: Public
Page 49/58

duration, the read field being stable for the prescribed amount of time, or the "when data available" condition becoming true. These correspond to the possible ways of specifying the end of an event cycle as defined

- ECReport
It represents a single report within an event cycle.
- ECReportGroup
It represents one group within an ECReport.
- ECReportGroupList
An ECReportGroupList Is included in an ECReportGroup when any of the four boolean fields includeEPC, includeTag, includeRawHex, and includeRawDecimal of the corresponding ECReportOutputSpec are true.
- ECReportGroupListMember
Each member of the ECReportGroupList is an ECReportGroupListMember and is constructed from information read from a single Tag.
- ECReportMemberField
Each ECReportMemberField within the fieldList of an ECReportGroupListMember gives the value read from a single field of a single Tag.
- ECReportGroupCount
It is included in an ECReportGroup when the includeCount field of the corresponding ECReportOutputSpec is true.
- ECTagStat
provides additional, implementation-defined information about each "sighting" of a Tag, that is, each time a Tag is acquired by one of the Readers participating in the event cycle.
- ECReaderStat
An ECReaderStat contains information about sightings of a Tag by a particular Reader.
- ECSightingStat
An ECSightingStat contains information about a single sighting of a Tag by a particular Reader.
- ECTagTimestampStat
It is a subclass of ECTagStat.

ALECallback Interface

The ALECallback interface is the path by which ALE Server delivers asynchronous results from event cycles to subscribers.

Whenever a transition specifies that "reports are delivered to subscribers" the ASPIRE's ALE Server attempt to deliver the results to each subscriber by invoking the callbackResults method of the ALECallback interface once for each subscriber, passing the ECReports for the event cycle, and using the binding and addressing information specified by the notification URI for that subscriber as specified in the subscribe call. All subscribers receive an identical ECReports instance.

ALE Logical Reader API (Licensed Accada/Fosstrak Implementation)

The ALE Logical Reader API is an interface of ALE Server through which clients may define logical reader names, each mapping to one or more sources/actuators provided by the implementation as described in the Filtering and Collections core specifications [2]. The API also allows the manipulation of configuration properties associated with logical reader names.

The Logical Reader API provides a standardized way for an ALE client to define a new logical reader name as an alias for one or more other logical reader names. The API also enables the manipulation of "properties" (name/value pairs) associated with a logical reader name. Finally, the API provides a means for a client to get a list of all the logical reader names available, and to learn certain information about each logical reader.

The commands that are implemented at the ASPIRE's ALE Server are:
- define
- update
- undefine
- getLogicalReaderNames
- getLRSpec
- addReaders
- setReaders
- removeReaders
- setProperties
- getPropertyValue
- getStandardVersion
- getVendorVersion

LRSpec
    It describes the configuration of a Logical Reader.
LRProperty
    A logical reader property is a name-value pair. Values are generically represented as strings in the Logical Reader API. The ALE implementation is responsible for any data type conversions that may be necessary.

ID: D3.3_Data Collection, Filtering and Application Level
Events.doc10
Revision: 0.10

Date: 29 Apr 2009

Security: Public
Page 51/58

ALE EventCycle (Licensed Accada/Fosstrak Implementation)

Overview

The "heart" of ALE's functionality is the EventCycle. An EventCycle is the smallest unit of interaction between an ALE client and an ALE implementation through the ALE reading API. In other words an EventCycle is an interval of time during which tags are read. This Section will provide an overview of the implementation of an EventCycle in the AspireRFID ALE.
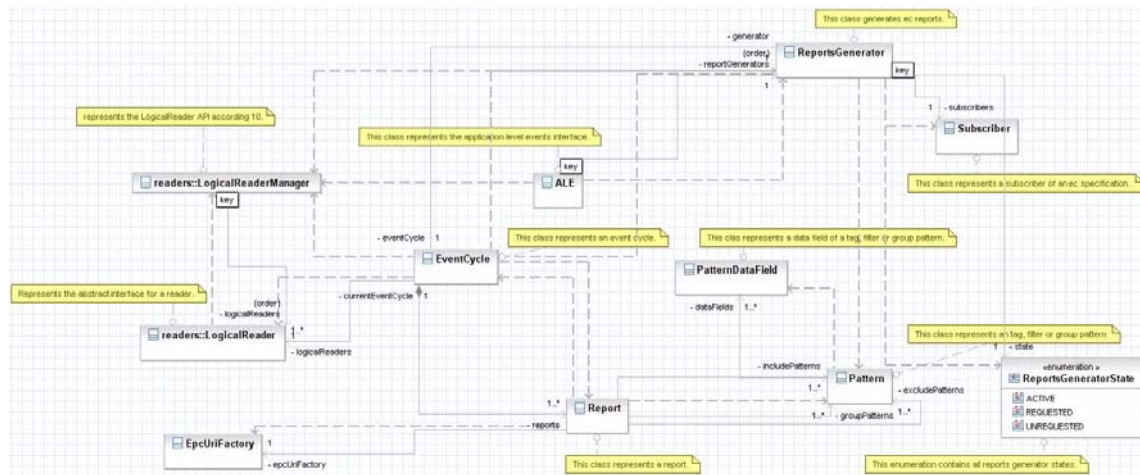

Figure 14 ASPIRE F&C Event Cycle Implementation UML

Implementation

The EventCycle is implemented as a thread. After the thread creation it waits on the EventCycles monitor. As soon as there are subscribers for the EventCycle the ReportsGenerator launches the EventCycle through the launch method and the EventCycle runs as long as specified by the duration value.

The reports are generated and sent through the ReportsGenerator Class as shown on the UML diagram above to the subscribers. After some cleanup and preparations for the next EventCycle, the EventCycle sets itself sleeping and waiting periods for the next launch call.

Life Cycle

An EventCycle is constructed according to an EventCycle specification (called ECSpec). The ECSpec specifies several parameters. The most important ones are the time interval during which tags are read and the readers where tags shall be collected.

Whenever a client defines a new EventCycle through the ALE interface, a new ReportsGenerator will be created along an EventCycle. The ReportsGenerator acts as a gateway to the EventCycle for clients. A client does not subscribe on an

EventCycle but on the associated ReportsGenerator and then the ReportsGenerator ensures that the EventCycle is started/stopped and that subscribers (clients) receive the resulting tags. Upon its creation the EventCycle acquires the readers from the logical reader API and registers as an Observer. Until now the EventCycle does not accept tags. When a client subscribes for an EventCycle, the ReportsGenerator starts the requested EventCycle and tags are now accepted by it. When the specified duration (EventCycle duration) is over, the EventCycle is stopped and the tags are passed through the Reports class where they are filtered and then distributed to the clients by the ReportsGenerator. In case where the EventCycle is not repeated (this means that one EventCycle does not run multiple times) all readers are deregistered and the EventCycle is destroyed together with its ReportsGenerator by a call to the stop() method. [1]

ALE Reader Interfaces Architecture

One important feature of the ASPIRE middleware platform is the ability to connect with multiple RFID readers. As Described in Deliverable D3.2 The ALE Server as shown in Figure 5 above supports two main reader protocols to connect with the various readers. These two protocols are the EPC-RP and The EPC-LLRP.

Low Level Reader Protocol Interface (AspireRfid Implementation)

The first communication interface between the ALE and an RFID reader is the EPC LLRP (Low Level Reader Protocol). This communication is feasible by using TCP protocol for both the notification channel over which XML LLRP reports are transferred from the reader directly to the server and for the reader operation programming by exchanging XML LLRP messages. Moreover the specific protocol supports capture of sensor and user tag data from EPCglobal UHF Gen2 tags which will be forwarded to the F&C server for processing.
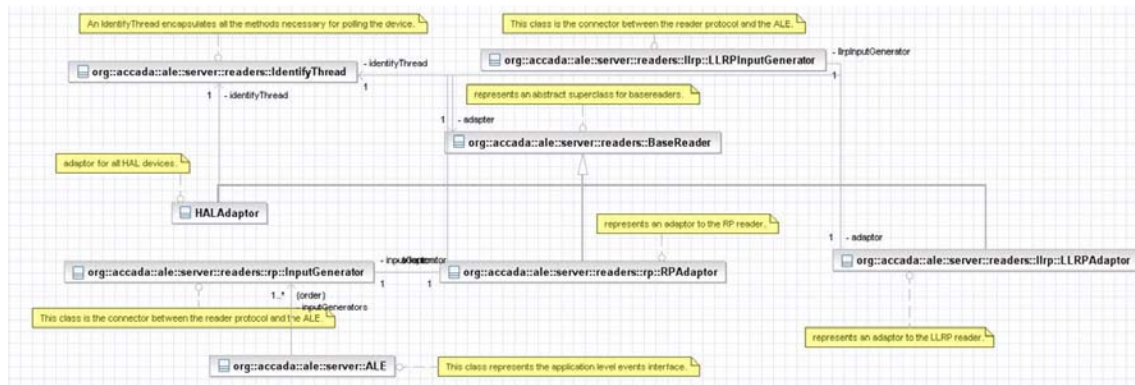

Figure 15 ASPIRE F&C Reader Interfaces Implementation UML

As shown in the UML diagram above (Figure 15**Error! Reference source not found.**) we are using the LLRPAdaptor class which extends the BaseReader. By overriding the following BaseReader's methods:
- start: start the reader
- stop: stop the reader
- connectReader: place the connection setup between the reader and the reader adaptor
- disconnectReader: destroy the connection between the reader and the adaptor
- identify: to poll the reader for the tags' report
- update: to update through the logical reader API for dynamic specification of the reader
- initialize: this method is used to setup the adaptor

And by using LLRP messages to implement them we achieve the communication between an LLRP Reader and the F&C Server.

LLRPInputGenerator which is used by the LLRPAdaptor creates a thread which undertakes the task of connecting with the reader, taking in consideration the logical reader specification file, and programming it by:

- Setting readers configuration,
- Adding Readers Operation Specifications
- Enabling them.

LLRPAdaptor starts the IdentifyThread which is polling the LLRP Reader (depending on the defined ECSpec's Boundary Specs) by starting the Reader Operation Specifications every time an application subscribes to the F&C server. The defined ECSpecs is using this LLRP reader for collecting the data it requests.

For the LLRP communication protocol we are using the LLRP toolkit library which houses the development of open source libraries in various languages to help reader and software vendors build and parse LLRP messages.

Reader Protocol Interface (Accada/Fosstrak Implementation)

The second communication interface between the ALE server and the RFID reader is the EPC-RP. This communication is feasible by using HTTP protocol for both the notification channel over which XML RP reports are transferred from the reader directly to the server and for the reader operation programming by exchanging XML RP messages. Moreover the specific protocol supports capture of sensor and user tag data from EPCglobal UHF Gen2 tags, which will be forwarded to the F&C server for processing.

As shown in the UML diagram above (**Error! Reference source not found.**) we are using the RPAdaptor class which extends the BaseReader. By overriding the following BaseReader's methods:

- start: start the reader
- stop: stop the reader
- connectReader: place the connection setup between the reader and the reader adaptor
- disconnectReader: destroy the connection between the reader and the adaptor
- identify: to poll the reader for the tags' report
- update: to update through the logical reader API for dynamic specification of the reader
- initialize: this method is used to setup the adaptor

And by using RP messages to implement them we achieve the communication between an RP Reader and the F&C Server.

RPInputGenerator which is used by the RPAdaptor creates a thread which undertakes the task of connecting with the reader, taking in consideration the logical reader specification file, and programming it by:

- Setting the Notification Channel Endpoint
- Creating a read trigger
- Creating a Notification Trigger
- Creating a Data Selector

- Adding the Notification Trigger to the Notification Channel.

RPAdaptor starts the Identify Thread which is polling the RP Reader (depending on the defined ECSpec's Boundary Specs) by getting all the read reports from the defined RP Reader Device sources every time an application subscribes to the F&C server and the defined ECSpecs is using that RP reader for collecting the data it demands.

For the RP protocol communication, we are using the FossTrak Reader Proxy which is a Java class library that supports the communication with a reader that implements the EPCglobal Reader Protocol Version 1.1.

ID: D3.3_Data Collection, Filtering and Application Level
Events.doc10
Revision: 0.10

Date: 29 Apr 2009

Security: Public
Page 56/58

APENDIX II ASPIRE ALE API Implementation Status

The following tables summarize the AspireRfid Filtering and Collection specifications Implementation status. The tables refers to the combination of Accada/FosTrak implementation, including the AspireRfid amendments.

Built-in Fieldnames, Datatypes, and Formats

Specification Fields	Implemented	Partially Implemented	Comments
Built-in Fieldnames		FC,RP,LLRP	
epc fieldname		FC,RP,LLRP	Not implemented for all EPC types
epcBank fieldname		FC,RP,LLRP	Implemented only for Gen2 Tag read command
tidBank fieldname		FC,RP,LLRP	Implemented only for Gen2 Tag read command
The afi fieldname		FC,RP,LLRP	Implemented At TDT
The nsi fieldname		FC,RP,LLRP	Implemented At TDT
Built-in Datatypes and Formats		FC,RP,LLRP	
epc datatype			
Binary Encoding and Decoding of the EPC Datatype			
EPC datatype Formats			
EPC datatype Pattern Syntax		FC	Not implemented for all EPC types
EPC datatype Grouping Pattern Syntax		FC	Not implemented for all EPC types

Table 2: Built-in Fieldnames, Datatypes, and Format

Reading API

Specification Fields	Implemented	Partially Implemented	Comments
ALE – Main API Class	FC,RP,LLRP		
Error Conditions		FC,RP,LLRP	
ECSpec		FC,RP,LLRP	
ECLogicalReaders	FC,RP,LLRP		
ECBoundarySpec		FC	start/stop trigger and duration not implemented,fix stableSetInterval
ECTime	FC		
ECTimeUnit	FC		
ECReportSpec		FC	filterSpec, includeSpecInReports and statProfileNames

			not Implemented
ECReportSetSpec	FC		
ECGroupSpec		FC	Fieldspec is not implemented
ECReport OutputSpec	FC		
Validation of ECSpecs		FC	
ECReports		FC	
ECInitiation Condition		FC	Trigger not implemented
ECTermination Condition		FC	Trigger, Data Available and Duration not implemented
ECReport	FC		
ECReportGroup	FC		
ECReportGroupList	FC		
ECReportGroup ListMember		FC	Stats not implemented
ECReport MemberField	FC		
ECReport GroupCount	FC		
ECTagStat		LLRP	
ECReaderStat		LLRP	
ECSightingStat		LLRP	
ECTag TimestampStat		LLRP	

Table 3: Reading API

Logical Reader API

Specification Fields	Implemented	Partially Implemented	Comments
Logical Reader API	FC,RP,LLRP		
Error Conditions	FC,RP,LLRP		
Conformance Requirements	FC,RP,LLRP		
LRSpec	FC,RP,LLRP		
LRProperty	FC,RP,LLRP		

Table 4: Logical Reader API

ID: D3.3_Data Collection, Filtering and Application Level
Events.doc10
Revision: 0.10

Date: 29 Apr 2009

Security: Public
Page 58/58