Collaborative Project

ASPIRE

Advanced Sensors and lightweight Programmable middleware for Innovative Rfid Enterprise applications

FP7 Contract: ICT-*215417*-CP

WP2 – Requirements and specifications

Public report - Deliverable

ASPIRE Middleware and Programmability Specifications

Due date of deliverable:	30/09/2008
Actual Submission date:	30/09/2008

Deliverable ID:	**WP2/D2.4**
Deliverable Title:	ASPIRE Middleware and Programmability Specifications
Responsible partner:	Athens Information Technology (AIT)
Main Contributors:	John Soldatos (AIT) Nektarios Leontiadis (AIT) Nikos Kefalakis (AIT) Ioannis Christou (AIT) Denis Ruffieux (MELEXIS) Didier Donsez (UJF) Lionel Touseau (UJF) Sofyan Mohammad Yousuf (OSI) Ramiro Samano Robles (IT) Panos Dimitropoulos (SENSAP)
Estimated Indicative Person Months:	12

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

Revision: 1.3
Dissemination Level: RE

Document Information

Document Name: **ASPIRE Middleware and Programmability Specifications**
Document ID: **WP2/D2.4**
Revision: **1.4**
Revision Date: **30 September 2008**
Author: **Athens Information Technology**
Security: **RE**

Approvals

	Name	Organization	Date	Visa
Coordinator	Neeli Rashmi Prasad	CTIF-AAU	30/09/2008	
Technical Coordinator	John Soldatos	AIT	30/09/2008	
Quality Manager	Thomas Christiansen	CTIF-AAU	30/09/2008	

Reviewers

Name	Organization	Date	Comments	Visa
Ramiro Samano Robles	IT	19/09/2008		
Humberto Moran	Open Source Innovation (OSI)	22/09/2008		
Mathieu DAVID	CTIF-AAU	23/09/2008		

Document history

Revision	Date	Modification	Authors
0.1	19 May 08	Table of Contents / Structure	John Soldatos
0.2	04 July 08	Added: Introduction, Middleware Specifications Draft	John Soldatos, Nektrarios Leontiadis, Nikos Kefalakis
0.3	14 July 08	Reader Access Specifications	Nektrarios Leontiadis
0.4	16 July 08	F&C Specifications	Nikos Kefalakis
0.5	25 July 08	Early High Level Description of the ASPIRE tools	John Soldatos
0.6	10 Sep 08	Information Sharing, BEG, Connector, IDE & Tools Specifications	Nikos Kefalakis
0.7	16 Sep 08	Appendix, Introduction, Additions,	John Soldatos, Nikos Kefalakis,

		Fine Tuning, Executive Summary, Aspire Reader Access API	Panos Dimitropoulos, Denis Ruffieux
0.8	17 Sep 08	Revisions, Conclusions, Version Sent to Reviewers	John Soldatos, Nikos Kefalakis
0.9	19 Sep 08	Major Reader Access and minor F&C, Information Sharing, BEG, Connector, IDE addition/changes	Nikos Kefalakis
1.0	22 Sep 08	Revisions based on OSI comments	John Soldatos
1.1	23 Sep 08	Addition of NFC input	Didier Donsez, Lionel Tuseau
1.2	24 Sep 08	Addition of input on ASPIRE Business Process Management	John Soldatos, Nikos Kefalakis, Ioannis Christou
1.3	29 Sep 08	Incorporation of comments from internal review, Revisions on Figures	John Soldatos, Nikos Kefalakis, Nektarios Leontiadis
1.4	30 Sep 08	Final Version	John Soldatos

Table of Contents

Executive summary

Among the main objectives of the ASPIRE project (http://www.fp7-aspire.eu) is to develop a royalty free RFID (Radio Frequency Identification) middleware platform, along with a range of tools that will enable the flexible and programmable implementation of RFID solutions. A key prerequisite for the implementation of such a platform and tools is the technical specifications of the software modules comprising the ASPIRE middleware and tooling. The purpose of this deliverable (namely ASPIRE D2.4 titled "ASPIRE Middleware and Programmability Specifications") is to illustrate these specifications in order to serve as a basis for their implementation in the ASPIRE RFID middleware platform. In devising these specifications the ASPIRE consortium has taken into account user requirements (established in document D2.2), middleware requirements and the ASPIRE architecture (illustrated in Deliverables D2.1 and D2.3), as well as the overall technical, technological and research goals of the project, as the later are reflected in the ASPIRE Description of Work (DoW) document.

The specifications contained within this deliverable will be reflected in the "AspireRfid" open source software (OSS) project, which the ASPIRE consortium has already setup as an OSS project of the OW2 community. Hence, specifications established in this document will be gradually implemented within the forge of the "AspireRfid" project, which is currently accessible at: http://forge.objectweb.org/projects/aspire/.Note that the specifications established in this document represent a superset of the features that will be implemented in the scope of the ASPIRE project. Being an OSS project, ASPIRE will endeavour to attract competent contributors outside the ASPIRE consortium. The specifications contained in this deliverable can therefore serve as a guideline for potential contributors. Note also that this document attempts to prioritize requirements and specifications on the basis of their criticality for the functionality of the ASPIRE middleware. Hence, core specifications that are absolutely essential for the deployment of RFID solutions are prioritized over other less important features.

We acknowledge that this deliverable cannot provide an exhaustive list of all the features that might be implemented/supported in the scope of the ASPIRE project. We cannot rule out the possibility of having this list extended in the future, especially in the case where external contributors will engage in the implementation, but also as more SME (Small Medium Enterprises) requirements are derived from the trials and/or the ASPIRE innovation management framework process.

The deliverable is structured on the basis of the ASPIRE architecture. The latter provides a framework for defining the various modules of the ASPIRE middleware platform. Moreover, it highlights programmability requirements, which are also addressed by the ASPIRE tools. The ASPIRE architecture identifies the following main middleware modules for the ASPIRE middleware platform:

- Middleware modules for virtualising/abstracting reader access i.e. enabling the ASPIRE platform to be flexible in supporting different reader vendors and types.
- Middleware modules for filtering and collection, which decouple the ASPIRE middleware platform from the physical readers' configurations and details, as well as from how tags are sensed and read. The filtering and collection middleware produces application level events.
- Middleware modules for generating business events in a configurable and automated fashion i.e. enabling the ASPIRE middleware to generate business events on the basis of reports produced by the filtering and collection modules.
- Middleware modules and repositories for storing and managing business events.
- Middleware modules acting as connectors to legacy IT (Information Technology) systems such as Enterprise Resource Planning (ERP) systems, Warehouse Management Systems (WMS), as well as corporate databases.

Note that some of the above modules are prescribed as EPC (Electronic Product Code) compliant modules i.e. ensuring compliance with a major set of RFID standards. This is particularly true for specifications relating to reader access and filtering. However, in-line with the ASPIRE architecture this deliverable introduces several middleware functions and tools that are not prescribed in existing standards. Specifically, the specifications contained in this deliverable specify the following innovative modules and tools:

- A business event generation (BEG) middleware module, which translates filtered reports into business events in an automatic fashion.
- Management modules enabling the end-to-end management of the whole RFID infrastructure, comprising both RFID hardware and middleware.
- A set of tools enabling business process management over the ASPIRE middleware.

Each of the above middleware modules is specified in fair detail in subsequent sections of this deliverable. Furthermore, special sections of the document are devoted to the ASPIRE tools comprising tools for configuring logical readers, managing filters, managing the RFID infrastructure, managing RFID enabled processes and ultimately supported integrated development of RFID applications. These tools will be also implemented in the scope of the AspireRfid project.

Overall, the deliverable describes the main capabilities and functionalities of the ASPIRE middleware and tools. We believe that these functionalities will result into environments that could ease RFID adoption by Small Medium Enterprises (SMEs), through facilitating development and deployment of RFID solutions. This is because the specifications provided in this deliverable consider not only technical requirements and standards, but also user requirements described in related ASPIRE Deliverable D2.2 and articulated by the SMEs themselves in the scope of the various "RFID Information Days" events of the ASPIRE project. Note that a special paragraph of this deliverable illustrates how the various middleware modules, address end-user requirements (particularly SME requirements).

In addition to providing a roadmap for the implementation of middleware and tools, this deliverables paves the ground way for to the specification and implementation of the programmability functionalities of the ASPIRE middleware. In particular, along with the implementation of the various tools, ASPIRE will endeavour to produce a domain specific language for describing fully fledged RFID solutions. The introduction of such a language can standardize the programmability capabilities of the ASPIRE middleware, which will enable third-parties to build ASRPIE compliant tools.

The tools and functionalities presented in this deliverable, will serve as a basis for deriving the RFID solution language. Specifically, with a list of middleware specifications at hand, programmability tasks in WP3 and WP4, can focus on a specification defining/declaring the structure and characteristics of an RFID solution. We envisage that the project will define a language comprising programmable constructs for defining RFID solutions. These programmable constructs will be defined based on the technical specifications established in this document. In particular, the programmable constructs must cater for the flexible configuration of all the features that are described in this document. The specification of the language falls in the scope of future deliverables.

1 Introduction

Among the main objectives of the ASPIRE project is to develop and deliver a lightweight, royalty-free, programmable, privacy friendly, standards-compliant, scalable, integrated and intelligent middleware platform that will facilitate low-cost development and deployment of innovative fully automatic RFID solutions. In the sequel we briefly discuss the above properties of the ASPIRE middleware, platform, while at the same time listing ASPIRE activities and deliverables that boost the realization of these properties:

- Royalty-free: ASPIRE will offer a licensing scheme enabling free use of its RFID developments. The ASPIRE consortium opts for open-source developments and has therefore collected and analyzed its needs in terms of Open Source Licensing. The ASPIRE middleware will be licensed under LGPL v2 (Lesser General Public License), as illustrated in ASPIRE Deliverable 3.1.
- Lightweight: Contrary to state-of-the-art commercial middleware platforms which subsume and rely on the functionality of a host of middleware and database services, the ASPIRE middleware will not be resource intensive. Please note that an in-depth state-of-the-art review of RFID middleware platform is contained in ASPIRE Deliverable D2.1, while the lightweight nature of the middleware has also been established as a requirement from SME communities in the scope of Deliverable D2.2 (dealing with end-user requirements). Note also that part (i.e. specific libraries) of the ASPIRE middleware, will be able to run over low-cost specialized microsystems, which possess RFID sensing, filtering and communication capabilities. These libraries will be developed in WP3 and WP5 of the consortium, which deal with the ASPIRE middleware platform implementation and the ASPIRE low-cost reader respectively.
- Programmable: The ASPIRE RFID middleware platform will provide solution developers and integrators with the opportunity of configuring simple solutions using solution templates and tools. The configuration process will involve minimal coding, or even no coding at all for simple solutions and/or applications.
- Intelligent: On top of RFID programmability, the ASPIRE RFID middleware platform will incorporate intelligence enabling context-analysis and reasoning over numerous sensors observations.
- Standards-Compliant: The ASPIRE RFID middleware developments will comply with existing RFID standards, starting from EPC standards (i.e. mainly on filtering and eventing) for both intra-enterprise and inter-enterprise applications development. Note that ASPIRE will leverage both EPC standards, as well as related EPC open-source developments.
- Scalable: The ASPIRE platform will be capable of supporting numerous massively distributed tags, as most likely required in realistic applications for the networked enterprise.
- Privacy-Friendly: The ASPIRE middleware will incorporate best practices (e.g., minimalist data generation, keeping tags no longer than required, ignoring tags that are out of an application's scope) relating to the development of privacy friendly middleware. Please refer to ASPIRE Deliverable D2.5, which elaborates on the privacy specifications of the middleware.

- Integrated: The ASPIRE platform will offer a complete integrated environment for specifications, development, integrations and experimentations of the RFID components and concepts through a concurrent innovation engineering framework. The exploitation of a novel innovation management framework in the scope of the ASPIRE framework is currently described and worked out in ASPIRE Deliverable D2.3.

We expect that this platform will significantly boost the adoption of RFID technology, especially for SME communities that wish to use RFID as an innovation vehicle.

The purpose of this deliverable is to provide specifications for the ASPIRE middleware platform and tools, as well as specifications for the programmability functionality of the platform. The ASPIRE middleware specifications are fully aligned with the architecture of the project, which is specified within Deliverable 2.3. In particular, middleware modules specified in the scope of this deliverable, are included in the overall picture of the ASPIRE middleware architecture. Hence, this deliverable complements D2.3. Specifically:

- Deliverable D2.3 provide the high-level structuring principles of the middleware modules that comprise the ASPIRE architecture, while
- Deliverable D2.4 specifies the lower-level details of the ASPIRE middleware modules and their interactions, as well as tools and programmability features of the ASPIRE middleware.

Note that the ASPIRE middleware platform takes into account related standards in the area of reader abstraction, filtering and data collection, as well as business events. Several specifications are therefore based on the enhancement of the existing standards (notable EPC standards). In these cases the present deliverable refers directly to the base standards, while also outlining the extensions. Nevertheless, even in the case of standards implementations we underline the specific functionalities to be supported by the ASPIRE middleware, since:

- In several cases some standards will only be partly implemented within ASPIRE. This is not a deviation from standards compliance since several standards include optional features.
- It is important to prioritize crucial functionalities for early implementation over others less important functionalities.

Furthermore, the present deliverable specifies middleware functionalities that are not prescribed by current standards. Prominent examples of such modules include:

- The Business Event Generation module, which facilitate the automatic and programmable generation of business events.
- The Business Process Management middleware framework, which allows execution of composite RFID-enabled business processes.
- The Connector modules undertaking the connection with legacy ICT systems.

In addition to the ASPIRE middleware specifications, this deliverable specifies programmability functionalities, as well as tooling. Programmability features aim at easing the configuration of ASPIRE solutions. The ASPIRE programmability functionality will offer to RFID developers and consultants the possibility to

deploy RFID solutions through entering high-level meta-data for a company (including the business context of its RFID deployments), rather than through writing significant amounts of low-level programming statements. At the same time programmability functionalities would also aim at treating personal data as specified by ePrivacy and other Data Protection Directives. For example, through algorithms that clean up unnecessary data and maintain principles of data quality, limitation, and conservation. To this end, this deliverable specifies a novel tool for privacy-friendliness auditing and enforcement.

At the heart of the ASPIRE programmability, lies an integrated way to specify company data, business process data, as well as middleware configuration metadata for the full range of components that comprise an RFID solution. In particular, ASPIRE programmability will be specified in the form of an XML-based (Extensible Markup Language) specification language, which is will be easily amendable by appropriate tools. Based on the above-mentioned specification language (which is a subject of future WP4 deliverables), the ASPIRE tools will provide opportunities for configuring, editing and deploying RFID solutions over the ASPIRE middleware platform. In addition to handling the ASPIRE programmability specifications in an integrated fashion, ASPIRE will also provide individual tools for configuring all the middleware modules of the platform (e.g., the filtering and collection module, the information sharing module, the reader access and virtualization module). All these tools will be bundled in a common Integrated Development Environment (IDE), providing access to all the ASPIRE tools. The functionality and the design of these tools are also specified in this document.

Please note that the ASPIRE consortium will open its developments to the open-source community. The ASPIRE consortium has initiated the "AspireRfid" project of the OW2 community, which will be accessible at: http://forge.objectweb.org/projects/aspire/, as a result of ASPIRE Deliverable D7.4.

It is envisaged that skilful community developers will actively engage in ASPIRE in order to implement the ASPIRE middleware platform and/or the ASPIRE Integrated Development Environment (IDE). Therefore, this deliverable should not be seen as an exhaustive list of functionalities that will be implemented in the scope of the project. On the contrary it is a specification of features that could be implemented either within the ASPIRE consortium or based on the involvement of the open-source community. This deliverable will therefore serve as a basis for starting and evolving a roadmap of ASPIRE Developments.

This deliverable is structured as follows:

- Section 2 introduces the Middleware Specifications and especially the relationships between each of its modules in the scope of the ASPIRE architecture. It also outlines how the different middleware blocks address identified SMEs requirements with respect to RFID deployment. Moreover, it illustrates how the ASPIRE middleware components can be used for deployments of varying scale and complexity.

- Section 3 sets out the various Reader Access Specifications which consists of the Hardware Abstraction Layer (HAL), the EPC-RP readers, the EPC-LLRP readers, the HF Readers, the ASPIRE Low-Cost Reader, as well as popular NFC readers (phones, mass-market products such as Nabaztag/tag, …). This section is influenced by related EPC reader access standards (notably EPC-RP, EPC-LLRP).
- Section 4 specifies the middleware module, which is of the top importance for RFID deployments, namely the Filtering and Collection Specifications. This includes specifications for filtering sensor streams, reading data from logical readers, writing data to tags, managing logical readers, as well as regulating access to readers and tags.
- Section 5 focuses on the Information Sharing Repository and Services Specification, which describes the ASPIRE Information Sharing repository as a repository of business events, along with a set of interfaces for accessing these events in both a synchronous and asynchronous fashion.
- Section 6 exhibits the Business Event Generation Specifications, which consists of operations that collect the static company data (also called master Data), process the delivered tag sequences, and ultimately populate the information sharing repository in a configurable and automatic fahsion. The section focuses also on the interfaces between the BEG and F&C modules.
- Section 7 introduces the Connector Specifications, which refers to the adapter framework for interfacing the RFID middleware system to the various corporate RDBMS (Relational Database Management Systems) and/or other enterprise systems (such as ERP (Enterprise Resource Planning) systems).
- Section 8 presents the ASPIRE IDE and Tools Specifications, which consist of the Integrated Development Environment, the management console tool, the readers configuration tool, the logical reader configuration tool, the filtering specifications editor, the F&C commands execution tool, the company/master data editor tool, the connector operations tool and finally the workflow management editor tool. In this section the novel ASPIRE business processes management concept is also introduced, along with an innovative privacy auditing tool.

Finally at the Appendix A we set out examples of business events for common elementary warehouse management processes such as receiving, moving within Logical Warehouses, order collection, pick & pack, order shipment and inventory. These events exemplify the ASPIRE business process management framework, through illustrating the notion of elementary RFID-enabled business processes, as well as how they can be combined into composite ones.

2 Middleware Specifications

2.1 Relationship to the ASPIRE Architecture

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

The high level architecture of the middleware is depicted in the following Figure 1. A figure depicting the main middleware building blocks of the architecture using UML 2.0 Component diagram notation is also provided (Figure 2).

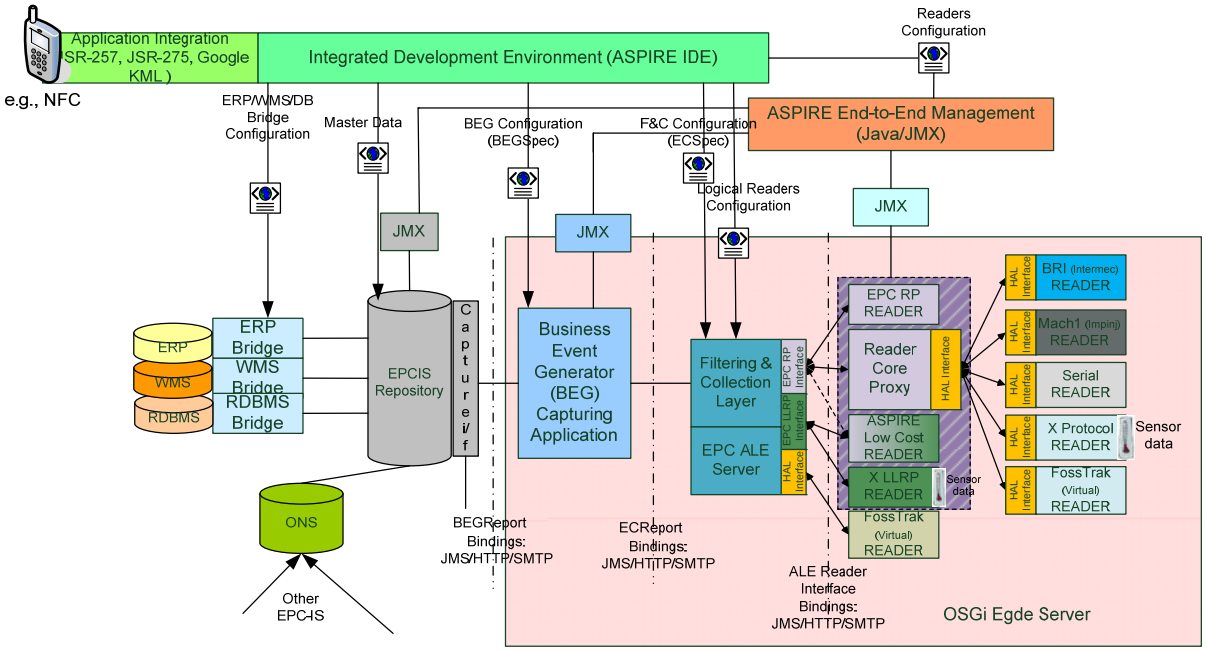

Figure 1: Overview of the ASPIRE Middleware Architecture

Each node in this figure represents a hierarchical level of functionality, starting from the hardware level, and provides a functional abstraction to a conceptually lower hierarchical level. Within each node, the corresponding specification that will be adopted is stated, whenever there is one available.

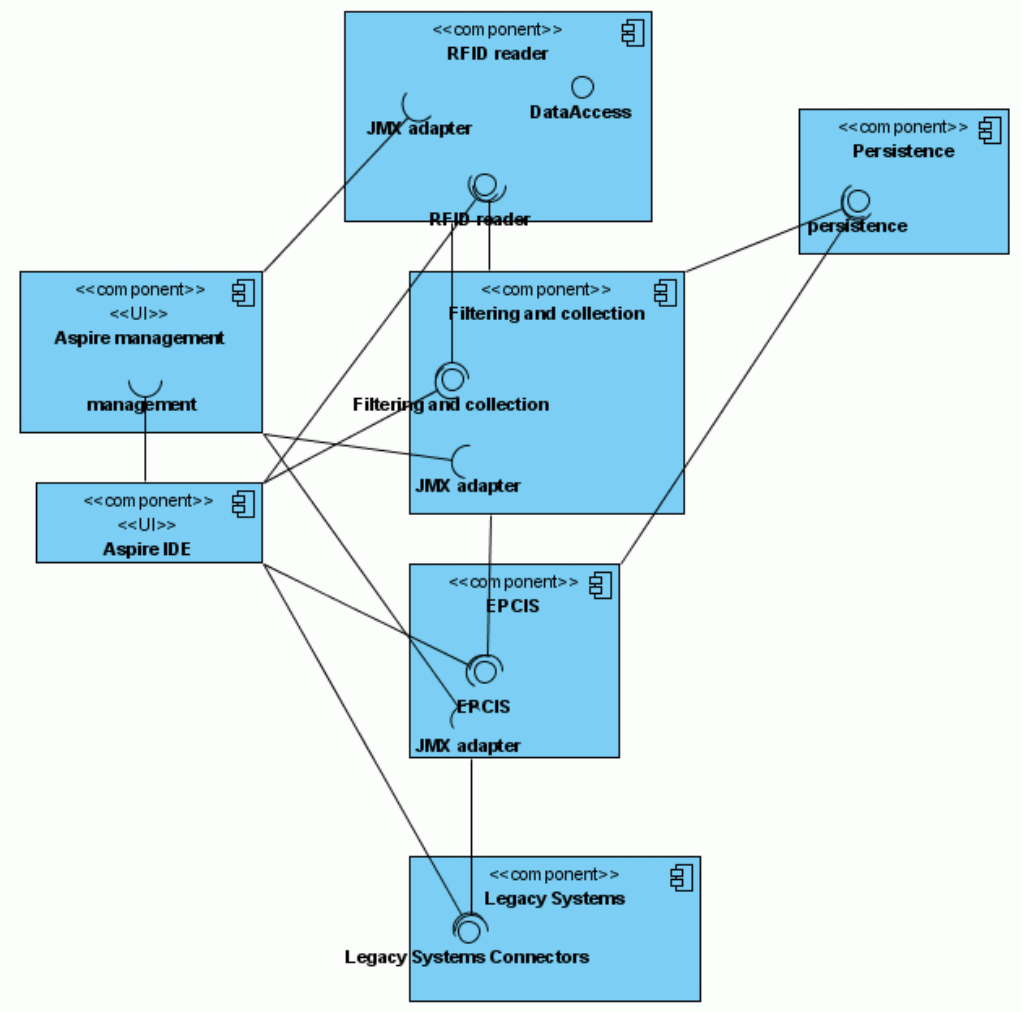

Figure 2: ASPIRE Components

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

The low level events though lesser than the raw RFID reads, they are significant in amount and do not provide high level – or business level – information. The role of this additional filtering and business eventing layer is handled by the Filtering and Collection and the Business Events Generator (BEG) component. These two components act in a complementary manner transforming the lower level events into business events. This transformation is only possible with the provision of additional metadata, which are appropriately handled by BEG.

This information (i.e. business events) is then forwarded to a higher hierarchical level, where it is consumed by the Information System level (IS). The IS level comprises a repository (i.e. a database) which aggregates events received by the

lower levels, applies additional business logic and stores business information, which could then be conveyed to the company's enterprise IT systems (e.g., Warehouse Management Systems (WMS), Enterprise Resource Planning (ERP) systems and corporate databases). Hence, well-defined connectors should drive the integration of information sharing repositories with enterprise business systems. The connectors are also in charge of exchanging information between the IS repositories and the business systems using data-centric (e.g., direct data access) and/or messaging mechanisms (e.g. EDI, XML messaging, Web Services) mechanisms. Moreover, in the scope of open loop systems this information can provided to other business partners, through either the enterprise systems or the information sharing repositories.

Apart from this functional plane of the architecture, there is management plane which manages and orchestrates the subsequent components. The management plane ensures that the middleware components comprising an ASPIRE system operate appropriately, while at the same time providing functionality for runtime management of the modules (e.g., starting, stopping, deploying and (re)configuring components).

Overall, the middleware specifications prescribed in this deliverable relate to and complement the ASPIRE architecture. The later drives the integration, interfacing and interaction of middleware modules in the scope of a unified and coordinated distributed software system. Note that ASPIRE architecture is thoroughly presented in ASPIRE Deliverable D2.3. An interim (preliminary) version of the ASPIRE architecture has already been provided in the scope of Deliverable D2.3a. Note that the architecture addresses the most common requirements for Automatic Identification Applications [10].

Under this prism, the following chapters, drill down to the various architectural nodes and specify their main functionalities. For each specification we provide a summary along with its reason of existence, while at the same time outlining its critically for the AspireRfid implementation roadmap.

2.2 ASPIRE Architecture vs. EPC Global Architecture

According to the ASPIRE Description of Work, the ASPIRE middleware will pursue compatibility with EPCglobal [17] architecture and related middleware specifications. Specifically, ASPIRE will take into account and pursue compliance with the EPC-RP [3], EPC-LLRP [4], EPC-ALE [2], [19] and EPC-IS [8] specifications. At the same time however, ASPIRE will introduce a range of unique middleware components and tools that are not prescribed in any EPCglobal specifications or other standards. These components are prescribed and implemented as part of the ASPIRE research work. Specifically, ASPIRE will develop the novel components that extend or are beyond the remit of EPC specifications:

- The ASPIRE business event generator a configurable component operating as a capture application that can turn raw RFID data to business events.
- The ASPIRE connectors architecture, which specified basic connectivity of the middleware platform with legacy IT applications (e.g., ERPs and databases).

- The ASPIRE management infrastructure and applications, which enables end-to-end management of all the servers and elements of the middleware platform.
- The ASPIRE Tools and Integrated Development Environment that enable visual end-to-end development and deployment of RFID solutions over the ASPIRE middleware platform.
- The ASPIRE Business process management framework, which allows for creating and managing RFID-enabled processes.

2.3 Classification of Middleware Specifications

The ASPIRE Middleware specifications can be clustered into the following categories:

- Specifications for reader access and reader virtualization, presented in Section 3. These specifications deal with vendor independent access to readers, which allows the ASPIRE middleware to be used with HF readers, UHF readers, from multiple vendors including the ASPIRE low-cost reader developed in WP5 of the project. Note also that deliverable D3.2 provides more information on the reader and tags virtualizations solutions currently implemented in the ASPIRE middleware.
- Specifications for filtering and collection, which decouples RFID applications from knowing the details of the physical layer, while also providing functionality for obtaining filtered RFID data. These specifications are presented in section 4.
- Specifications for business events generation, which add business context to filtered RFID sensor streams. The specifications are presented in section 6.
- Specifications for information sharing, which describe the operation and functionalities of the ASPIRE business events repositories are presented in section 5.

Each one of these categories is analyzed in a distinct chapter. Following the specification of the main middleware modules, we also provide programmability specifications (as part of the ASPIRE tools), which we link them to the capabilities of the ASPIRE IDE.

2.4 Relationship to End-Users (SMEs) Requirements

By and large the specifications detailed in this deliverable must lead in the implementation of a middleware platform that could greatly facilitate end-users in general and SMEs in particular to easily deploy RFID solutions. To this end, the presented specifications take into account SME's requirements illustrated in end-user workshops ("information days"), as well as Deliverable D2.2, as follows:

End-users SMEs Requirement	Related ASPIRE Middleware or Tool Specification
Lightweight Nature	Exploitation of lightweight containers (OSGi and Spring illustrated in deliverable D2.3b)
Integration with Legacy Systems	ASPIRE Connectors Specification (see Connector Specifications), enabling the ASPIRE interfacing with legacy IT systems and corporate databases
Lower Integration Effort	ASPIRE Tools and Business Process Management (see

and Consulting Costs	ASPIRE IDE and Tools Specifications), to facilitate development, deployment and integration
Bar-Code Support	Part of ASPIRE Tag Data Translation Implementation (see Deliverable D3.2)
Reader Vendor Independence	Specification and implementation of ASPIRE reader access specifications (see Reader Access Specifications section and Deliverable D3.2).
Minimal Maintenance Costs	Specification of ASPIRE end-to-end management functionality (see Management Console Specifications)
Secure Access to RFID functionalities	F&C Access Control API (see Filtering and Collection Specifications)
Management of SME Business Processes; Innovating with RFID	ASPIRE Business Process Management Framework and Related Workflow Tools (see sub-section Business Process Management and Workflow Management Editor for Composite Business Processes)
Privacy Friendliness (especially for consumer related deployment)	ASPIRE Privacy Tools (see ASPIRE IDE and Tools Specifications)

Table 1: SME requirements and related ASPIRE middleware or Tools Specifications

Note that the above SME requirements are addressed not only in D2.4, but also in other related deliverables of the ASPIRE project.

2.5 Middleware Building Blocks and ASPIRE applications

Among the main objectives of the ASPIRE project is to produce a generic and configurable middleware platform, which could support a multitude of RFID deployments, with particular emphasis on SME related deployments. Hence, the middleware specified in this deliverable is designed to support RFID applications, pilots and deployments of varying functionality and scale.

As a primary target the ASPIRE middleware blocks defined in this document are not aimed at supporting large scale “open loop” systems and deployments, similar to those developed and trialled by Wall-Mart and the U.S Department of Defense (DoD). Rather ASPIRE’s primary target are smaller scale solutions covering a wide range of asset tracking and inventory management scenarios, as well as other ROI (return-on-investment) generating case studies. These target case studies focus on very specific business problems, which an RFID enabled system, can solve even within a single enterprise. A main characteristic of these smaller scale deployments is that tracking, traceability and identification occur within a warehouse or a single supply chain. Note that these smaller scale solutions are in-line with most RFID vendor initiatives worldwide, which are gradually refocusing their strategies in order to address both large scale deployment and smaller-scale opportunities. ASPIRE envisages that small applications (i.e. closed loop islands) could one day become integrated into larger scale open loop systems.

Depending on the scale and target goals of their RFID deployment middleware developers and RFID consultants should prioritize the adoption and use of

ASPIRE middleware modules based on the scale of the target application. Large scale open loop solutions must pay emphasis on implementing the full range of middleware components described in this deliverables. On the other hand smaller scale closed loop systems must prioritize the Filtering and Collection (F&C), reading and tag virtualization components (defined in ASPIRE D3.2). Moreover, for some very simple systems our experience shows that custom filters over a reader access solution for the target hardware could provide a rapid and acceptable solution. Table 2 presents the middleware building blocks that we envisaged as mandatory for various application categories.

Application Type / Middleware Block	**HAL (see Reader Access Specifications)**	**Reader Access (see Reader Access Specifications)**	**F&C (see Filtering and Collection Specifications)**	**Business Events (see Information Sharing Repository and Services Specification, Business Event Generation Specifications**
Simple	Yes	Recommended	Recommended	No
Simple Closed Loop	Yes	Yes	Recommended	No
Complex Closed Loop	Yes	Yes	Yes	Recommended
Open Loop	Yes	Yes	Yes	Yes

Table 2: ASPIRE Middleware Building Blocks for various application categories

3 Reader Access Specifications

3.1 Overview

The ASPIRE hardware components are the main sources of data of an ASPIRE middleware system. These hardware components will mainly be RFID readers and possible other auxiliary sensors (e.g., for sensing physical quantities). The specifications defined in this section deal with the interfacing to RFID hardware components and make the ASPIRE middleware capable of communicating using standardized messages. In particular, ASPIRE is reader agnostic through supporting the following specifications:

- EPCglobal EPC-RP (Reader Protocol)
- EPCglobal EPC-LLRP (Low-level Reader protocol)

In addition we provide information about the way non-compliant devices communicate with the ASPIRE middleware and about the methods of interaction between the middleware and the ASPIRE Low Cost reader (which is prototyped in WP5 of the project).

Note that we capitalize on existing standards for solving the reader virtualization problem (i.e. to achieve hardware vendor independence), as also illustrated in Deliverable D3.2.

3.2 Hardware Abstraction Layer (HAL)

The role of this layer is to unify the way the ASPIRE middleware interacts with the RFID readers from multiple vendors that support varying protocols. This is based on the introduction of a hardware abstraction layer (HAL) and the provision of a fixed instruction set to upstream middleware layers which consume RFID readings from the hardware [12].

Specifications that satisfy the need for a norm at this level are the EPCglobal Reader Protocol (RP), the EPCglobal Lower Level Reader Protocol (LLRP). These protocols define the standard bindings through witch an application can send messages in a standardized format.

The methods of communication between the HAL and the hardware itself vary, depending on the hardware vendor and it may require a serial connection, an Ethernet connection, etc. The protocols of communication may also vary from a raw TCP (Transmission Control Protocol) connection, to SSL (Secure Sockets Layer) and HTTP (Hypertext Transfer Transport Protocol). The same will apply for the command and message encodings, which may be text, XML or binary.

The following figure gives an overview of the HAL architecture.

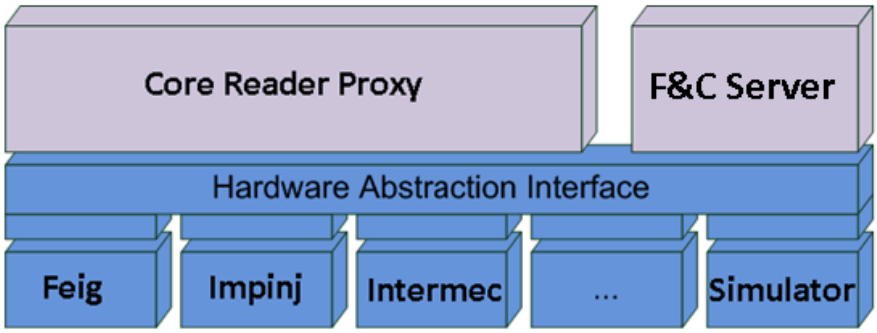

Figure 3: HAL Architectural Overview

The Hardware Abstraction (Figure 4) defines the interface between the HAL, the F&C server, the Reader Core (3.2.1) or any other application using the HAL. It standardises access to various readers and simulators of readers. This allows uniform usage. The readers and simulators become interchangeable because the code specific to the reader is part of the HAL and not of the application. The implementations of the Hardware Abstraction interface are divided into multiple modules, one for the simulators and one for each reader manufacturer. A module can contain one or multiple reader controllers.

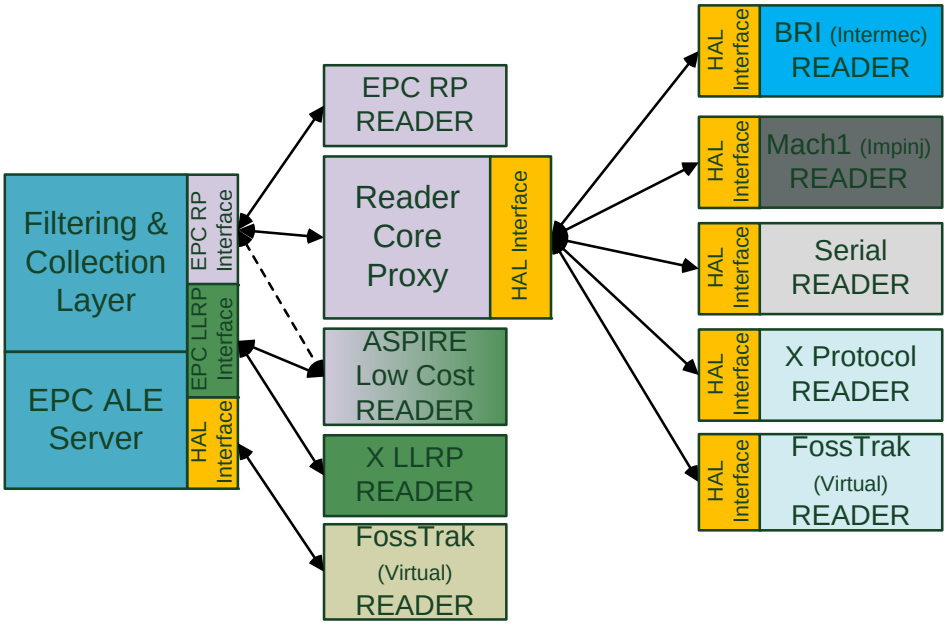

Figure 4: ASPIRE HAL Connections

3.2.1 Core Reader

In order to transform non EPC Reader Protocol readers into compliant readers we are using a core reader application, which is used (as shown in Figure 4) as a mediator between a reader supporting protocol “X” and the corresponding F&C Reader Protocol Interface. By deploying the appropriate HAL module at the Reader Core we make whatever reader compliant to RP. Every reader with an implementation of the Hardware Abstraction interface can be controlled over the Reader Protocol.

The core reader support TCP and HTTP for transporting reader protocol messages, while the message content can be either XML or Text. In addition, it support synchronous and asynchronous messaging (through the reader's protocol Notification Channels mechanisms). Furthermore, support for other reader protocol characteristics, such as triggers, data selectors must be provided.

3.2.2 EPC-RP Support

The Reader Protocol standard [4] is defined by EPCglobal and provides a high-level interface of interaction between a middleware and a compliant RFID device. The *interaction* is defined by the combination of the formatting of the exchanged messages with the underlying communication protocol used to exchange these messages. The parameters of the interaction are defined through a standardized handshaking procedure each time a communication channel needs to be established. The EPCglobal Reader Protocol standard version 1.1 will be supported by the Aspire middleware [4]. It have been currently implemented in the scope of Deliverable D3.2 and will be incorporated in the AspireRfid codebase.

3.2.3 EPC-LLRP Support

The Low Level Reader Protocol standard [3] is also defined by EPCglobal and provides a low level interface for interaction between a middleware and a compliant RFID device. It is called low-level because it provides control of RFID air protocol operation timing and access to air protocol command parameters. The design of this interface recognizes that in some RFID systems, there is a requirement for explicit knowledge of RFID air protocols and the ability to control Readers that implement RFID air protocol communications. It also recognizes that coupling control to the physical layers of an RFID infrastructure may be useful for the purpose of mitigating RFID interference.

LLRP is an application layer protocol and does not provide retransmission, or reordering facilities. State consistency between the Client and the Reader is critical for the correct functioning of the system. Using LLRP messages, the Client updates the Reader state, which includes Reader configuration parameters, dynamically created data structures (e.g., ROSpecs, AccessSpecs, etc), and possibly vendor-defined data. For this reason, LLRP requires acknowledgements for the client to Reader transactions – this provides a fail-safe mechanism at the LLRP layer to cope with network error situations. Also, to cope with intermittent connections, a Client can request a Reader's configuration state to confirm that a Reader's state is consistent with the Client after the Client reconnects. The Reader-to-Client messages are primarily reports, status notifications or keep-alives. The Aspire middleware will support the EPCglobal Low Level Reader Protocol standard version 1.0.1 [3]. It have been currently implemented in the scope of Deliverable D3.2 (based on the LLRP toolkit [9]) and will be incorporated in the AspireRfid codebase.

3.2.4 NFC Reader Support

NFC Forum[1] defines specifications for Near-Field Communications (NFC)[2] applications such as product information, smart posters, discount vouchers, ticketing and payment. The specification covers only the list of supported RFID tags standards and products[3] and the information stored in the tags. The forum has not yet defined an architecture (similar to the EPCglobal) one to integrate NFC information in company information systems. JCP has specified an API for contactless communications (JSR 257[4]). This API enables to develop J2ME applications using NFC tags.

Since the NFC had already met the mass market in Japan[5] and will probably meet it in developed countries[6][7] in the next years, NFC phones should be supported by the Aspire middleware. NFC phones could be integrated in Aspire as readers compliant with the EPCGlobal Reader Protocol standard version 1.1. Moreover, NFC phones could query an ONS (Object Naming Service) implementation using Web services or RESTFul services.

3.3 ASPIRE Low-Cost Reader Specifications

The RFID reader that is being developed in the project is low-cost and equipped with lightweight middleware. It will provide appropriate subsets of the RP and LLRP protocols. The Aspire middleware will interact with this device through this interface and will exchange standardized messages. In particular, the ASPIRE reader/middleware interface (for the low-cost reader) will support LLRP, RM (Reader Management), DCI (Discovery Configuration and Initialization), and optionally RP messages, within the context of the EPC standards. The main procedures (messages) of each protocol that have an influence on the desired interface are listed in the sequel:

- The ASPIRE Low-cost reader will support the following Low level reader protocol primitives:
  - GET_READER_CAPABILITIES
  - GET_READER_CAPABILITIES_RESPONSE
  - ADD_ROSPEC
  - ADD_ROSPEC_RESPONSE
  - DELETE_ROSPEC
  - DELETE_ROSPEC_RESPONSE
  - START_ROSPEC
  - START_ROSPEC_RESPONSE
  - STOP_ROSPEC
  - STOP_ROSPEC_RESPONSE
  - ENABLE_ROSPEC

[1] http://www.nfc-forum.org
[2] http://java.sun.com/developer/technicalArticles/javame/nfc/
[3] ISO/IEC 14443A, ISO/IEC 14443 B, Sony FeliCa.
[4] http://jcp.org/en/jsr/detail?id=257 is led by Nokia which delivers it in its J2ME SDKs
[5] *20 millions FeliCa phones mi-2007 and 40 millions mi-2008*
[6] Frost & Sullivan (March 07) : "One third of all mobile phones will be NFC-equipped in a span of three to five years"
[7] Strategy Analytics (September 06): "Mobile phone-based contactless payments will facilitate over $36 billion of worldwide consumer spending by 2011".

- ENABLE_ROSPEC_RESPONSE
- DISABLE_ROSPEC
- DISABLE_ROSPEC_RESPONSE
- GET_ROSPECS
- GET_ROSPECS_RESPONSE
- ADD_ACCESSSPEC
- ADD_ACCESSSPEC_RESPONSE
- DELETE_ACCESSSPEC
- DELETE_ACCESSSPEC_RESPONSE
- ENABLE_ACCESSSPEC
- ENABLE_ACCESSSPEC_RESPONSE
- DISABLE_ACCESSSPEC
- DISABLE_ACCESSSPEC_RESPONSE
- GET_ACCESSSPECS
- GET_ACCESSSPECS_RESPONSE
- CLIENT_REQUEST_OP
- CLIENT_REQUEST_OP_RESPONSE
- GET_REPORT
- RO_ACCESS_REPORT
- KEEPALIVE
- KEEPALIVE_ACK
- READER_EVENT_NOTIFICATION
- ENABLE_EVENTS_AND_REPORTS
- ERROR_MESSAGE
- GET_READER_CONFIG
- GET_READER_CONFIG_RESPONSE
- SET_READER_CONFIG
- SET_READER_CONFIG_RESPONSE
- CLOSE_CONNECTION
- CLOSE_CONNECTION_RESPONSE
- CUSTOM_MESSAGE

Figure 5 presents an example about LLRP support in the ASPIRE low-cost reader.

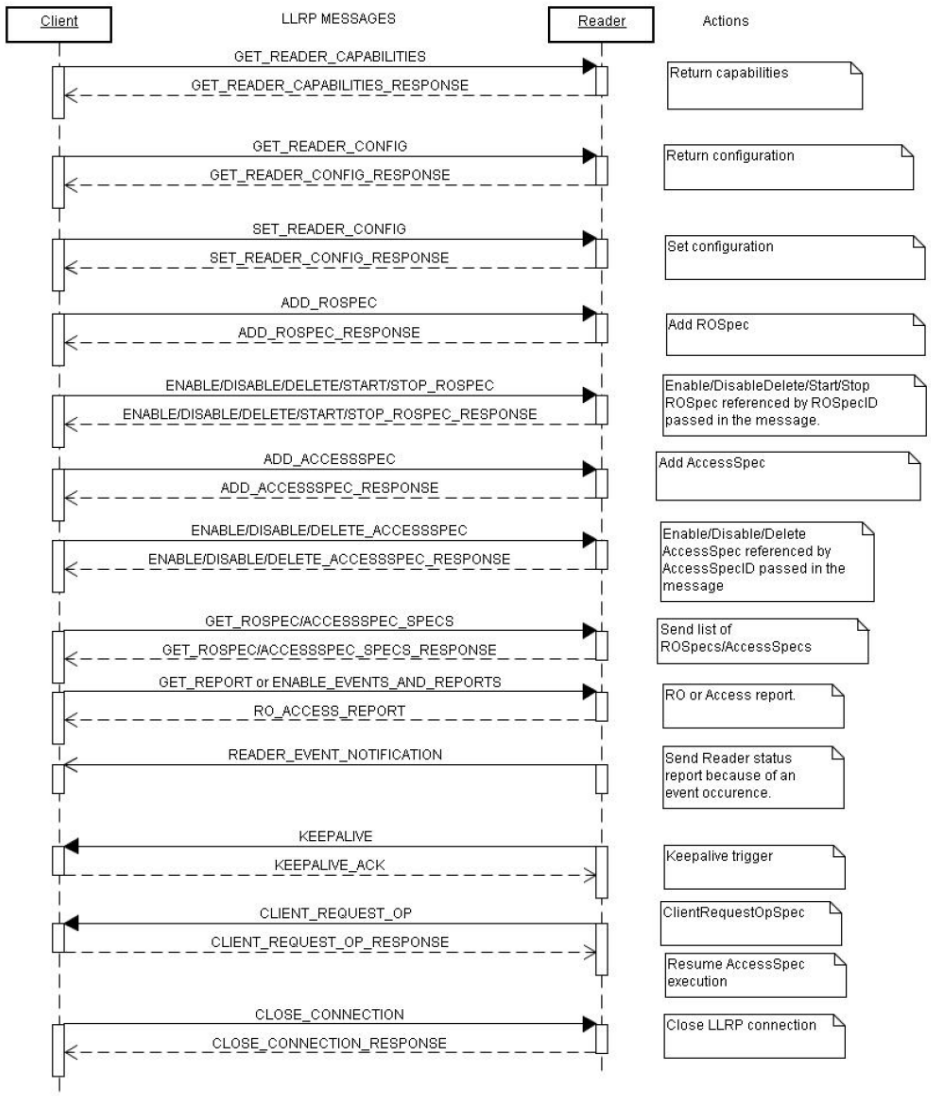

Figure 5: Example of LLRP procedures to be supported by the ASPIRE low-cost reader

- The ASPIRE Low-cost reader will support the following groups of Reader Management primitives:
  - ReaderDevice
  - NotificationChannel
  - AlarmChannel
  - ReadPoint
  - AntennaReadPoint
  - Source Object
  - Trigger Object
  - IOPort Object
  - EdgeTriggeredAlarmControl
  - TTOperationalStatusAlarmControl

- The ASPIRE Low-cost reader will support the following groups of the Reader protocol primitives:
  - Object ReaderDevice
  - Object Source

- Object TagFieldValue
- Object ReadPoint
- Object Trigger
- Object TagSelector
- Object CommandChannel
- Object NotificationChannel
- Object DataSelector
- Enumeration Objects
- Object TagField

As already outlined the low-cost reader interface will also provide support for the emerging DCI (Discovery Configuration and Initialization) standard.

3.4 Reader Access Specifications Requirements overview

The following tables summarizes the ASPIRE Reader Access specifications, which are selected and prioritized among the multitude of specifications detailed in [3] and [4]. Implementation priority values from 1 to 5 with 5 being of most importance. Note that these values concern the relative implementation priorities within the ASPIRE project.

1.1	EPCglobal RP specification [4] requirements (prioritized for implementation in AspireRfid)	Applied to	Priority
1.1.1	Reader Device interface	Reader Core, F&C and Low-cost reader	4
1.1.2	Reader Device commands	Reader Core, F&C and Low-cost reader	4
1.1.3	Source interface	Reader Core, F&C and Low-cost reader	4
1.1.4	Source commands	Reader Core, F&C and Low-cost reader	4
1.1.5	Triggers interface	Reader Core, F&C and Low-cost reader	3
1.1.6	Trigger commands	Reader Core, F&C and Low-cost reader	3
1.1.7	Tag Selectors interface	Reader Core, F&C and Low-cost reader	3
1.1.8	Events interfaces	Reader Core and F&C	4
1.1.9	Notification and Command Channels interfaces	Reader Core, F&C and Low-cost reader	4
1.1.10	Notification Channel commands	Reader Core, F&C and Low-cost reader	4

1.1.11	Data Selectors interface	Reader Core, F&C and Low-cost reader	4
1.1.12	Data Selector commands	Reader Core, F&C and Low-cost reader	4
1.1.13	Tad Selector commands	Reader Core, F&C and Low-cost reader	4
1.1.14	Tag Fields and Tag Field Values interfaces	Reader Core, F&C and Low-cost reader	4
1.1.15	Tag Field and Tag Field Value commands	Reader Core, F&C and Low-cost reader	4
1.1.16	Read Point interface	Reader Core, F&C and Low-cost reader	4
1.1.17	Read Point commands	Reader Core, F&C and Low-cost reader	4
1.1.18	EventType enumeration interface	Reader Core, F&C and Low-cost reader	4
1.1.19	TriggerType enumeration interface	Reader Core, F&C and Low-cost reader	4
1.1.20	FieldName enumeration interface	Reader Core, F&C and Low-cost reader	4
1.1.21	PredefinedTagFieldName enumeration interface	Reader Core, F&C and Low-cost reader	2
1.1.22	Error Handling	Reader Core and F&C	4
1.1.23	standard message and Transport Bindings	Reader Core, F&C and Low-cost reader	3

Table 3: ASPIRE middleware specifications for EPC-RP support

1.2	**EPCglobal LLRP specification requirements[3] (prioritized for implementation in AspireRfid)**	**Applied to**	**Priority**
1.2.1	GET_READER CAPABILITIES	Reader Core, F&C and Low-cost reader	4
1.2.2	GET_READER CAPABILITIES RESPONSE	Reader Core, F&C and Low-cost reader	4
1.2.3	ADD_ROSPEC	Reader Core, F&C and Low-cost reader	4
1.2.4	ADD_ROSPEC_RESPONSE	Reader Core, F&C and Low-cost reader	4

1.2.5	DELETE_ROSPEC	Reader Core, F&C and Low-cost reader	4
1.2.6	DELETE_ROSPEC RESPONSE	Reader Core, F&C and Low-cost reader	4
1.2.7	START_ROSPEC	Reader Core, F&C and Low-cost reader	4
1.2.8	START_ROSPEC RESPONSE	Reader Core, F&C and Low-cost reader	4
1.2.9	STOP_ROSPEC	Reader Core, F&C and Low-cost reader	4
1.2.10	STOP_ROSPEC RESPONSE	Reader Core, F&C and Low-cost reader	4
1.2.11	ENABLE_ROSPEC	Reader Core, F&C and Low-cost reader	4
1.2.12	ENABLE_ROSPEC RESPONSE	Reader Core, F&C and Low-cost reader	4
1.2.13	DISABLE_ROSPEC	Reader Core, F&C and Low-cost reader	4
1.2.14	DISABLE_ROSPEC RESPONSE	Reader Core, F&C and Low-cost reader	4
1.2.15	GET_ROSPECS	Reader Core, F&C and Low-cost reader	4
1.2.16	GET_ROSPECS RESPONSE	Reader Core, F&C and Low-cost reader	4
1.2.17	ADD_ACCESSSPEC	Reader Core, F&C and Low-cost reader	4
1.2.18	ADD_ACCESSSPEC RESPONSE	Reader Core, F&C and Low-cost reader	4
1.2.19	DELETE_ACCESSSPEC	Reader Core, F&C and Low-cost reader	4
1.2.20	DELETE_ACCESSSPEC RESPONSE	Reader Core, F&C and Low-cost reader	4
1.2.21	ENABLE_ACCESSSPEC	Reader Core, F&C and Low-cost reader	4
1.2.22	ENABLE_ACCESSSPEC RESPONSE	Reader Core, F&C and Low-cost reader	4

1.2.23	DISABLE_ACCESSSPEC	Reader Core, F&C and Low-cost reader	4
1.2.24	DISABLE_ACCESSSPEC RESPONSE	Reader Core, F&C and Low-cost reader	4
1.2.25	GET_ACCESSSPECS	Reader Core, F&C and Low-cost reader	4
1.2.26	GET_ACCESSSPECS RESPONSE	Reader Core, F&C and Low-cost reader	4
1.2.27	CLIENT_REQUEST_OP	Reader Core, F&C and Low-cost reader	4
1.2.28	CLIENT_REQUEST_OP RESPONSE	Reader Core, F&C and Low-cost reader	4
1.2.29	GET_REPORT	Reader Core, F&C and Low-cost reader	4
1.2.30	RO_ACCESS_REPORT	Reader Core, F&C and Low-cost reader	4
1.2.31	KEEPALIVE	Reader Core, F&C and Low-cost reader	4
1.2.32	KEEPALIVE_ACK	Reader Core, F&C and Low-cost reader	4
1.2.33	READER_EVENT NOTIFICATION	Reader Core, F&C and Low-cost reader	4
1.2.34	ENABLE_EVENTS_AND REPORTS	Reader Core, F&C and Low-cost reader	4
1.2.35	ERROR_MESSAGE	Reader Core, F&C and Low-cost reader	4
1.2.36	GET_READER_CONFIG	Reader Core, F&C and Low-cost reader	4
1.2.37	GET_READER_CONFIG RESPONSE	Reader Core, F&C and Low-cost reader	4
1.2.38	SET_READER_CONFIG	Reader Core, F&C and Low-cost reader	4
1.2.39	SET_READER_CONFIG RESPONSE	Reader Core, F&C and Low-cost reader	4
1.2.40	CLOSE_CONNECTION	Reader Core, F&C and Low-cost reader	4

1.2.41	CLOSE_CONNECTION RESPONSE	Reader Core, F&C and Low-cost reader	4
1.2.42	CUSTOM_MESSAGE	Reader Core, F&C and Low-cost reader	4

Table 4: ASPIRE middleware specifications for EPC-LLRP support

1.3	EPCglobal RM specification requirements	Applied to	Priority
1.3.1	Reader Device interface	Reader Core and Low-cost reader	3
1.3.2	Reader Device commands	Reader Core and Low-cost reader	3
1.3.3	Source interface	Reader Core and Low-cost reader	3
1.3.4	Source commands	Reader Core and Low-cost reader	3
1.3.5	Notification Channel interface	Reader Core and Low-cost reader	3
1.3.6	Notification Channel commands	Reader Core and Low-cost reader	3
1.3.7	Alarm Channel interface	Reader Core and Low-cost reader	3
1.3.8	Alarm Channel commands	Reader Core and Low-cost reader	3
1.3.9	Read Point interface	Reader Core and Low-cost reader	3
1.3.10	Read Point commands	Reader Core and Low-cost reader	3
1.3.11	Antenna Read Point interface	Reader Core and Low-cost reader	3
1.3.12	Antenna Read Point commands	Reader Core and Low-cost reader	3
1.3.13	Triggers interface	Reader Core and Low-cost reader	3
1.3.14	Trigger commands	Reader Core and Low-cost reader	3
1.3.15	IO Port interface	Reader Core and Low-cost reader	3
1.3.16	IO Port commands	Reader Core and Low-cost reader	3
1.3.17	Alarm Control interface	Reader Core	3
1.3.18	Alarm Control commands	Reader Core	3
1.3.19	Edge Trigger Alarm Control interface	Reader Core and Low-cost reader	3
1.3.20	Edge Trigger Alarm Control commands	Reader Core and Low-cost reader	3
1.3.21	TT Operational Status Alarm Control interface	Reader Core and Low-cost reader	3
1.3.22	TT Operational Status Alarm Control commands	Reader Core and Low-cost reader	3
1.3.23	Alarm interface	Reader Core	3
1.3.24	Alarm commands	Reader Core	3

1.3.25	Free Memory Alarm interface	Reader Core	3
1.3.26	Free Memory Alarm commands	Reader Core	3
1.3.27	Failed Write Alarm interface	Reader Core	3
1.3.28	Failed Write Alarm commands	Reader Core	3
1.3.29	Failed Erase Alarm interface	Reader Core	3
1.3.30	Failed Erase Alarm commands	Reader Core	3
1.3.31	Failed Kill Alarm interface	Reader Core	3
1.3.32	Failed Kill Alarm commands	Reader Core	3
1.3.33	Failed Lock Alarm interface	Reader Core	3
1.3.34	Failed Lock Alarm commands	Reader Core	4
1.3.35	Failed Mem Read Alarm interface	Reader Core	4
1.3.36	Failed Mem Read Alarm commands	Reader Core	4
1.3.37	TT Oper Status Alarm interface	Reader Core	4
1.3.38	TT Oper Status Alarm commands	Reader Core	4
1.3.39	Reader Device Oper Status Alarm interface	Reader Core	4
1.3.40	Reader Device Oper Status Alarm commands	Reader Core	4
1.3.41	IO Port Oper Status Alarm interface	Reader Core	4
1.3.42	IO Port Oper Status Alarm commands	Reader Core	4
1.3.43	Read Point Oper Status Alarm interface	Reader Core	4
1.3.44	Read Point Oper Status Alarm commands	Reader Core	4
1.3.45	Source Oper Status Alarm interface	Reader Core	4
1.3.46	Source Oper Status Alarm commands	Reader Core	4
1.3.47	Notification Channel Oper Status Alarm interface	Reader Core	4
1.3.48	Notification Channel Oper Status Alarm commands	Reader Core	4
1.3.49	Administrative Status enumeration interface	Reader Core	4
1.3.50	Administrative Status enumeration commands	Reader Core	4
1.3.51	Operational Status enumeration interface	Reader Core	4
1.3.52	Operational Status enumeration commands	Reader Core	4
1.3.53	Edge Triggered Alarm Direction enumeration interface	Reader Core	4
1.3.54	Edge Triggered Alarm Direction enumeration commands	Reader Core	4
1.3.55	Alarm Level enumeration interface	Reader Core	4
1.3.56	Alarm Level enumeration commands	Reader Core	4
1.3.57	Error Handling	Reader Core	4
1.3.58	standard message and Transport Bindings	Reader Core	4
1.3.59	Enable vendor extensions	Reader Core	4

Table 5: ASPIRE middleware specifications for EPC-RM support

4 Filtering and Collection Specifications

4.1 Overview

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

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

The Filtering and Collection 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 project, the Business Event Generation (BEG) middleware (described later in this deliverable) 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 filtering & collection middleware specification is influenced by the EPC specification [19]. 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 specific already 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 the data arriving from different readers in a report that is sent according to a pre-determined schedule to the subscribed applications. Since the middleware receives data from multiple readers, it provides specific filtering functionality depending on the different already defined specifications. So redundant events from different readers observing the same location are not included to the despatched report accomplishing the reduction of filtering and aggregation required to the registered application interpreting the captured RFID data.

The ASPIRE middleware must implement two interfaces between the filtering & collection middleware and upstream layers (i.e. business event generation modules or host applications). In particular:

- One interface is needed for transporting the RFID data. The TCP/HTTP protocol could therefore be adopted to this end. The transport interface must enable BEG and/or applications to receive RFID data either in a "push" or in "pull" fashion. 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 the 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 from all the ASPIRE management and development tools, which will need to configure and/or program the F&C server operation.

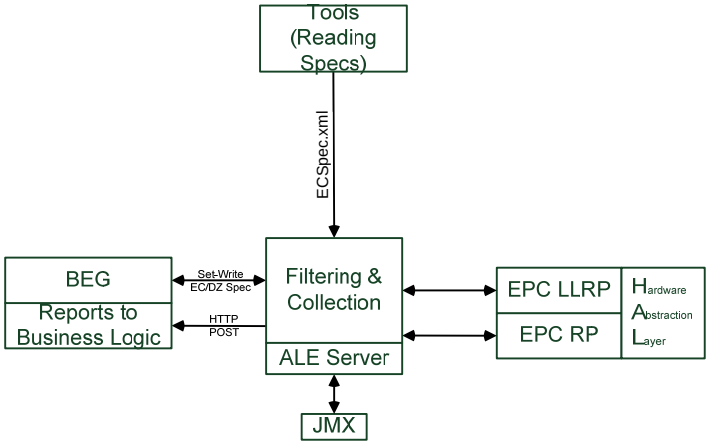

Figure 6 Filtering and Collection (ALE)

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.

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 of which describes 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.

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 provide useful information about the event cycle.
The ALE interface revolves around client requests and the corresponding reports that are produced. Requests can either be:

- *immediate*, in which information is reported on a one-time basis at the time of the request; or
- *recurring,* 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.

The available request modes are shown at the pictures below:

- Subscribe Mode: Asynchronous reports from a standing request

Figure 7 Asynchronous reports from a standing request (according to [19])

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

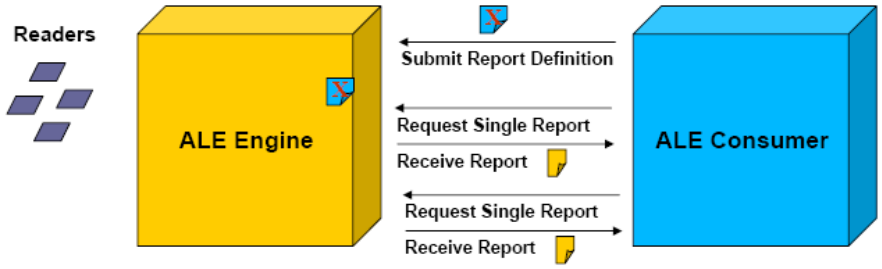

Figure 8 On-demand report from a standing request [19]

- Immediate Mode: Synchronous report from one-time request

Figure 9: Synchronous report from one-time request [19]

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

4.2 F&C Specifications

Several of the Filtering and Collection specifications presented here represent selected information from the Application Level Events (ALE) Specification Version 1.1 (for more information see [2]). This is because the EPC-ALE fulfils most of the requirements established in the previous paragraph. Note also that ASPIRE must pursue compliance with the EPC specification [2], in accordance to the project's Annex 1 ("Description of Work").

In the sequel we elaborate filtering and collection specifications in the following areas:

- Supported Fields, Data types and Formats, which specify the formatting of the rfid tag information, which will flow through the F&C middleware layer. The formulation of this information is important in order to support common tags and data formats (notably EPC and ISO related tag information).
- Accessing, configuring and managing the tags memory based on appropriate interfaces and APIs.
- Reading tags and returning tag steams according to a reading specification that defines how event cycles are calculated. Tags that are read will be returned in the form of appropriate reports that correspond to reading specifications.
- Writing tags, as required by several RFID applications that are concerned with printing tags.
- Managing logical readers (i.e. combinations of one or more physical readers and their antennas), which insulates the F&C layer from the details of the readers physical configuration.
- Access control to the F&C functionalities, in order to implement/ensure authorized access to the ASPIRE F&C operations.
- F&C Management, enabling management applications to manage the F&C server.

The following table (Table 6) depicts the classification of F&C functionalities in the above areas, also outlining some characteristic use cases where they are needed. It is evident that the specified F&C layer addresses several key requirements and use cases of Automatic identification applications.

F&C Specification Class	Key Functionalities	Sample Use Cases
Fields, Formats, Data types	-	Eases applications to support common tags
Tag Memory	Accessing Tag Memory	Reading User Defined Fields on an RFID Tag.
Reading Tags	• Definition of Event Cycles (e.g., when to read). • Definition of filters and groups. • Synchronous and Asynchronous access to data.	• Reading tags (e.g., items/products) of a specific category (based on a given pattern). • Repot Reading Differences (e.g.,, products removed from a shelf)
Writing API	• Definition of writing	Writing tags (e.g., in

	command cycles.	manufacturing and production) applications.
Logical Readers Management	• Defining logical readers based on combinations of physical readers	Changing the physical readers configuration to achieve grater accuracy (e.g., adding antennas and readers on a gate) without changing the reading specification i.e. in a way transparent to the upsteam applications
Access Control to F&C	• Authorized access to reading & filtering operations	Prohibit unauthorized applications to define reading specifications and execute/operate them on the F&C server.

Table 6: High-Level Classification of F&C Specifications and Associated Use Cases

Specifications in the above areas follow in the paragraphs below.

4.2.1 Supported Fieldnames, Data types, and Formats

The ASPIRE middleware will support the following fieldnames, data types, and formats.

Fieldnames

The ASPIRE middleware must support the following fieldnames:

- The epc fieldname, which might be exchanged in string format between the reader access layer and the F&C middleware module.
- The killPwd fieldname, in the scope of an F&C layer's interaction with a Gen2 Tag.
- The accessPwd fieldname, in the scope of an F&C layer's interaction with a Gen2 Tag.
- The epcBank fieldname, denoting the content of the EPC memory bank in EPC implementations. In non-ECP impelentation the epcBank fieldname could be exploited for other uses.
- The tidBank fieldname, denoting the content of the TID memory bank. For non EPC comliant tags this field can also be exploited for other uses.
- The userBank fieldname, denoting the content of the User memory bank in EPC Gen2 implementations. For non EPC Gen2 compliant tags this field can also be exploited for other uses.
- The afi field name denoting the offset 18h to 1Fh in the EPC/UII memory bank of a Gen2 Tag, which may hold the ISO 15962 Application Family Identifier (AFI). When interacting with a Gen1 Tag, an ALE implementation SHALL interpret the afi fieldname as a "field not found". When interacting with any other type of Tag, the interpretation of the afi fieldname is implementation dependent.
- The nsi fieldname, denoting the offset 17h to 1Fh in the EPC/UII memory bank of a Gen2 Tag, which holds the Numbering System Identifier (NSI).

When interacting with a Gen1 Tag, an ALE implementation SHALL interpret the nsi fieldname as a "field not found". When interacting with any other type of Tag, the interpretation of the nsi fieldname is implementation dependent.

- Generic Fieldnames, in order to support Absolute Address Fieldnames, Variable Fieldnames, as well as Variable Pattern Fieldnames.

Data types and Formats

In general, the specification of each data type in the ASPIRE F&C middleware indicates which formats may be used with that data type. Each format denotes whether it is permissible in both reading and writing contexts or only in reading contexts. A format must define the syntax for literal values, for filter patterns, and for grouping patterns. The following formats must be supported:

- The epc data type, refers to the space of values defined in the EPCglobal Tag Data Standard.
- The Unsigned Integer (uint) Datatype, having as space the set of non-negative integers.
- The bits Data type, referring to the set of all non-empty and finite-length sequences of bits.
- The ISO 15962 String Data type, referring to the iso-15962-string.

Extensions for sensors data

More and more, RFID applications requirements include the processing of data collected from sensors attached to the tagged objects (i.e. Active RFID) or associated with the objects environment (i.e. temperature and position of the transportation container, hygrometry of the storing warehouse, shocks or position (horizontal or vertical) during transportation …). Reports (e.g., ECReports) can be extended by custom data "a priori" ignored by the standard F&C by EPCglobal.

The Aspire Middleware should collect sensors data and add them in ECReports as custom extensions. BEG should include filtering rules taking into account collected sensors data (i.e. trig alerts if blood products are over heated …). Those sensors data should be stored and retrieved by the ASPIRE Information Sharing layer and displayed in the UI (e.g., curves, maps, etc.) (see section Information Sharing Repository and Services Specification). Sensor data should be collected from APIs such as OSGi WireAdmin, JCP JSR 256 (Mobile Sensor API) [23], JCP JSR 179 (Location API for J2ME) [24] and more. Sensors data should be convert and represented as well-known and standard representations such as OSGi position and measurements, JSR 275 (Units Specification) [25], WG84, ISO 31-0 (Quantities and units).

4.2.2 Accessing/Configuring/Managing Tag Memory

An API for managing user-defined fieldnames that refers to fixed-length, fixed-offset fields must be provided. User-defined fieldnames are equivalent in functionality to the absolute fixed address fieldnames or to the variable fieldnames. This API must support:

- Tag memory specifications, as a means to defining fieldname. In particular, the ASPIRE middleware should allow users to define their own fieldnames.

- Management of the Tag Memory based on appropriate tag memory specifications.
- Unordered lists of fieldnames, each fieldname mapping to a specific fixed field described by a bank, offset, and length.
- Unordered list of fieldnames, each fieldname mapping to a specific ISO 15962 data set named by an object identifier (OID).
- Single fixed-length field, as well as variable fields allowing upstream middleware layers (i.e. ALE clients) to associate symbolic names with an ISO 15962 object identifiers.

4.2.3 Reading Tags - Reading API

The ASPIRE middleware should provide an API for:

- Defining/specifying/configuring/managing reading specifications, including how an event cycle is to be calculated.
- Managing/Accessing Reports, which contain one or more reports generated from an activation of a reading specification.
- Both synchronous and asynchronous access to RFID data. Asynchronous data access should be supported based on appropriate data access interface.

Reading Specifications

A reading specification specifies an event cycle and one or more reports (i.e. lists of RFID tags) that will 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, and a list of specifications each of which describes a report to be generated from this event cycle. Hence a reading specification must include:

- An unordered list that specifies one or more logical readers that are used to acquire tags.
- A specification of how the beginning and end of event cycles are to be determined.
- A span of time measured in physical time units.
- The units of physical time that may be used in the scope of a timing specification.
- A URI that used to specify a start or stop trigger for an event cycle or command cycle.
- A list of reports that will results after the execution of an event cycle. As already specified, the filtering report must contain one or more reports. Whenever an event cycle completes, a list of reports will be generated, unless suppressed.
- A specification denoting the set of Tags is to be considered for filtering and output. Depending on the application this set can be:
    - All Tags read in the current event cycle.
    - Additions from the previous event cycle.
    - Deletions from the previous event cycle.
- A specification of the filtering to be carried out, which will specify the tags are to be included in the final report. In addition the particular tags that must be reported should also be specified (e.g., an application may be concerned for particular tag fields only).
- A specification of filtering patterns on the various tag fields.

- A specification of how tags should be group together in the scope of a report (in case grouping is needed).
- A specification of statistics profile that could be included in the resulting list of reports.

Filtered Reports

The output from an event cycle must be a list of reports. The list should contain one or more reports each one corresponding to a report specified within a filtering specification. In addition to the list of reports should obtain a number of metadata fields that provide useful information about the event cycle. Overall a filtered report must include:

- An indication of what kind of event caused the event cycle to initiate (e.g., an explicit start trigger, the expiration of the repeat period, or a transition to the requested state in the case where no start triggers were specified in the filtering specification).
- An indication of what kind of event caused the event cycle to terminate (e.g., the receipt of an explicit stop trigger, the expiration of the event cycle duration, the read field being stable for the prescribed amount of time, or the fact that data become available).
- A grouping of the reports, as well as mechanisms for grouping relevant reports.
- Statistical Information about "sightings" of a tag, in particular:
    - 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.
    - Information about sightings of a Tag by a particular Reader.
    - Information about a single sighting of a Tag by a particular reader.
    - Information according to application/user defined statistics profiles.

Callbacks and Asynchronous Reading

The ASPIRE F&C middleware implementation must deliver asynchronous results from event cycles to subscribers. Whenever a transition specifies that "reports are delivered to subscribers" the ASPIRE implementation SHALL attempt to deliver the results to each subscriber through an appropriate interface. The latter must include the reports corresponding to the event cycle, and direct them to the URI corresponding to the each subscriber.

4.2.4 Writing to Tags - ALE Writing API

Several RFID processes involve the tagging of items (e.g., during manufacturing and production (see [22])). Hence, apart from reading operations the ASPIRE middleware must also support writing operations.

The F&C middleware module should provide the means for managing command cycles, which are to writing, exactly what event cycles are to readings. The writing process of the ASPIRE F&C middleware should support:

- Specification of command cycles (i.e. how a command cycle is to be carried out). This specification should be defined in the scope of command cycle specifications.
- Synchronous and asynchronous execution of command cycles, with results sent in the form of command reports. Asynchronous execution should take place based on an appropriate callback mechanism.
- Reports containing a list of individual report instances, which are produced as a result of an activation of a command cycle.
- An unordered list of name/value pairs, each specifying a parameter name and a corresponding parameter value. Parameter values are string data that provide specific values to be used in tag commands.

Command Specification

A command specification should contain:

- One or more logical reader names;
- A boundary specification that identifies an interval of time;
- One or more command specifications that specify operations to be performed on a population of Tags visible to the specified logical readers during the specified interval of time.

The command specifications also imply what information is included in a report generated from each command cycle generated from this specification. Hence, command specifications must also specify:

- How the beginning and end of command cycles are to be determined.
- An inclusive/exclusive filtering to be applied over sets/populations of tags
- Ordered lists of one or more operation specifications, each of which describes a single operation to be performed on a tag. Operations include reading a field, writing a field, and other Tag operations.
- Statistics profiles to be included in the resulting command reports.
- Mechanisms for validating the command cycle specifications.

Command Reports

Command reports constitute the output of a command cycle. Each report contains an ordered list of individual command report instances, each one corresponding to report specification that has been associated with the command cycle specification. In addition to the reports instances themselves, command reports contains metadata fields that provide useful information about the command cycle. In particular a command report should include:

- A description of how a command cycle was started and ended.
- A description of what happened during the processing of a single tag.
- Information on the result of a single operation executing on a single tag during a command cycle.
- The possible outcomes for a given operation
- Additionally, 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 command cycle.

Writing tags

When writing tags the F&C implementation should:

- Allow upstream layers (e.g., BEG generator, applications) to formulate and populate the writing comments. To this end they may keep track of in-memory lists of tag values, along with patterns of tag values.
- Support random number generators (RNG), as a source of random numbers that can be used by the write commands.
- Delivers asynchronous results from command cycles to subscribers, through appropriate callbacks that deliver the command reports to subscribers through the notification URIs associated with them.

4.2.5 Managing Logic Readers - ALE Logical Reader API

The ASPIRE F&C middleware must allow definition, management and configuration of logical readers. The later are required to insulate the ASPIRE middleware from knowing the low-level details of physical readers and antennas configurations. Hence, the F&C middleware must support an interface through which clients may define logical reader names for use with the reading and writing operations specified above. This interface should also allow the manipulation of configuration properties associated with logical reader names.

We conveniently call this logical reader interface, logical reader API. The Logical Reader API should:

- Provide a way for F&C clients to define a new logical reader name as an alias for one or more other logical reader names.
- Cater for manipulating “properties” (name/value pairs) associated with a logical reader name.
- Provide a means for a client to get a list of all of the logical reader names that are available, and to learn certain information about each logical reader.
- Provide error handling capabilities.
- Describes the configuration of a Logical Reader, along with its properties (as name-value pairs).
- Be configurable in a way that can reduce the appearance of tags moving in and out of a reader’s field of view due to intermittent tag reads. This is similar to the tag smoothing mechanism introduced in [2].

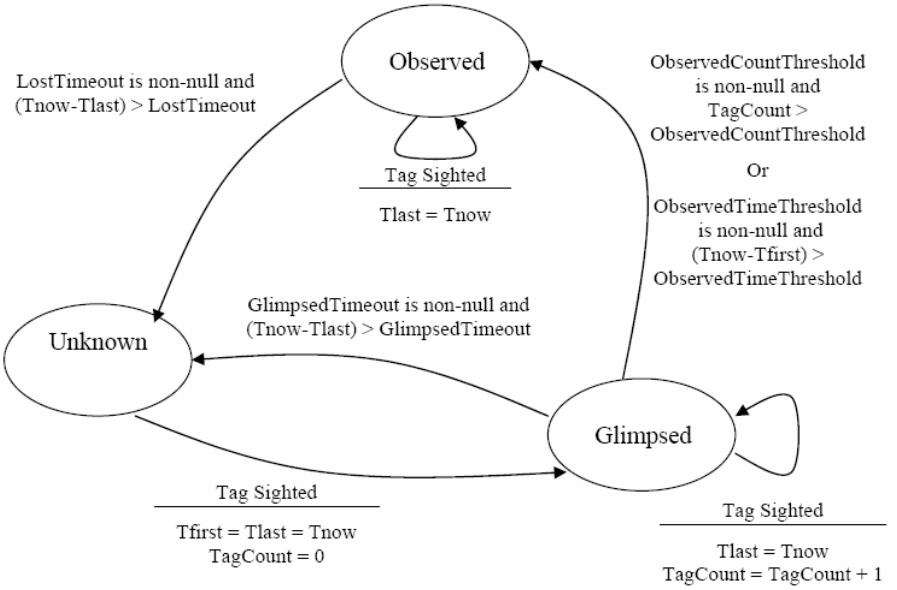

Figure 10: Tag smoothing finite state machine diagram (according to [2]).

4.2.6 Access Control to F&C Functionalities

The ASPIRE F&C middleware must also support an API for controlling client to the F&C functionalities and APIs (e.g., the reading, writing and logical readers' configuration) outlined above. This API should provide role-based way to associate access control permissions with client identities.

An authentication mechanism must also be in place at the bindings' layer. This authentication mechanism grants an identity, to each of the clients. A client identity should maps to one or more roles, which accordingly maps to one or more permissions. Each of the permissions denotes an access privilege to some F&C functionality (or API). Hence, the F&C client will be permitted to access those functionalities (i.e. sets of permissions) that are foreseen by its role.

Permissions are described by granting access to specific resources. A client may access only the resources for which it has access permission. The Access Control API specifies facilities that may overlap or conflict with facilities provided by the environment in which other ALE APIs are provided. For example, it is common in enterprises to centralize information about identities, roles, and permissions in repositories such as LDAP servers, so that this information may be shared across many different applications. In such a setting, it may not be appropriate for the system component including an ALE implementation to provide its own API for manipulating client identities and permissions, but instead defer to the mechanisms provided by the LDAP environment.

4.2.7 ALE Management

The F&C implementation should be manageable through external manager entities, in particular JMX entities. Hence, the ASPRIE F&C module will be a JMX-enabled management application. As shown in Figure 14 of the JMX Architecture [15] three layers should be implemented:

- Instrumentation Level.

- Agent Level.
- Adaptors level.

For every resource that needs management and monitoring the instrumentation process will be implemented. Java objects known as MBeans following the design patterns and interfaces defined in the JMX specification will be used for each and one of them to expose the management information in the form of attributes and operations and offer access to the instrumentation of resources. MBeans for the following functions may be created:

- Starting the various components of the F&C server
  This will mainly be achieved by bundlizing the various components to work within an OSGI container.
- Stopping the various components of the F&C server
  This will mainly be achieved by bundlizing the various components to work within an OSGI container.
- Managing the Logical readers
- Managing the Incoming reports
- Managing the Outgoing reports
- Managing the defined LRSpecs
- Managing the defined reading specifications (e.g., ECSpecs).
- Managing the Subscribers

Also an MBeanServer will be created which will contain the list of MBeans registered with it. All management operations performed on the MBeans will be done through the MBeanServer. All the JMX agents that will provide the set of services will reside at the MBeanServer. Each of these services is termed an agent service.

The JMX agent should contain at least one protocol adaptor or connector. These protocol adaptors and connectors provide the possibilities of remote management, by defining the manager components which are capable of communicating with the agents. Protocol adaptors and connectors make the agent accessible from remote management applications. They provide a view through a specific protocol of the MBeans instantiated and registered in the MBean server. For exporting JMX API instrumentation to remote applications Remote Method Invocation (RMI) will be used.

4.3 ASPIRE ALE API Specifications Requirements overview

The following table summarizes the ASPIRE Filtering and Collection Specifications. Implementation priority values from 1 to 5 with 5 denoting the most important priority.

4.3.1 Fieldnames, Data types, and Formats

C/N	Specification	Priority
3.1	**Support for Fieldnames (according to EPC-ALE)**	
3.1.1	Epc fieldname	4
3.1.2	killPwd fieldname	2
3.1.3	accessPwd fieldname	2
3.1.4	epcBank fieldname	2

3.1.5	tidBank fieldname	2
3.1.6	userBank fieldname	2
3.1.7	The afi fieldname	2
3.1.8	The nsi fieldname	2
3.1.9	Generic Fieldnames	2
3.1.10	Absolute Address Fieldnames	2
3.1.11	Variable Fieldnames	3
3.1.12	Variable Pattern Fieldnames	3
3.2	**Data types and Formats**	
3.2.1	EPC data type	4
3.2.2	Binary Encoding and Decoding of the EPC Data type	3
3.2.3	EPC data type Formats	3
3.2.4	EPC data type Pattern Syntax	3
3.2.5	EPC data type Grouping Pattern Syntax	3
3.2.6	Unsigned Integer (uint) Data type	3
3.2.7	Binary Encoding and Decoding of the Unsigned Integer Data type	2
3.2.8	Unsigned Integer Data type Formats	2
3.2.9	Unsigned Integer Pattern Syntax	2
3.2.10	Unsigned Integer Grouping Pattern Syntax	2
3.2.11	The bits Data type	2
3.2.12	Binary Encoding and Decoding of the Bits Data type	2
3.2.13	Bits Data type Formats	2
3.2.14	ISO 15962 String Data type	4
3.2.15	ISO 15962 String Format	4

Table 7: Specifications for Fieldnames, Data types and Formats

4.3.2 Tag Memory Specification API (According to EPC ALE)

C/N	Specification	Priority
3.3.1	Tag memory specifications, as a means to defining fieldname. In particular, the ASPIRE middleware should allow users to define their own fieldnames.	3
3.3.2	Management of the Tag Memory based on appropriate tag memory specifications.	3
3.3.3	Unordered lists of fieldnames, each fieldname mapping to a specific fixed field described by a bank, offset, and length.	3
3.3.4	Unordered list of fieldnames, each fieldname mapping to a specific ISO 15962 data set named by an object identifier (OID).	3
3.3.5	Single fixed-length field, as well as variable fields allowing upstream middleware layers (i.e. ALE clients) to associate symbolic names with an ISO 15962 object identifiers.	3

Table 8: Tag Memory Specification API

4.3.3 Reading API

C/N	Specifications	Priority
3.4.1	Support for Reading Specification: 1. Event cycles boundaries 2. Reports, 3. Logical readers 4. Filters, groups, patterns 5. Tags to be considered for filtering and output (all tags, additions, deletions) 6. Statistics Profile	5

3.4.2	Support for Filtered Reports 1. List of reports 2. Indication of what kind of event caused the event cycle to initiate Indication of what kind of event caused the event cycle to terminate 3. Grouping of the reports, as well as mechanisms for grouping relevant reports. 4. Statistical Information about “sightings” of a tag	5
3.4.3	Callbacks and Asynchronous Reading - Delivery of asynchronous results from event cycles to subscribers.	5

Table 9: Tag Reading Specifications

4.3.4 Tag Writing Specification

C/N	Specification requirements	Priority
		3
3.5.1	Command Support: 1. One or more logical reader names; 2. Boundary specification that identifies an interval of time; 3. Operations to be performed on a population of Tags visible to the specified logical readers during the specified interval of time. 4. How the beginning and end of command cycles are to be determined. 5. An inclusive/exclusive filtering to be applied over sets/populations of tags 6. Ordered lists of one or more operation specifications, each of which describes a single operation to be performed on a tag. Operations include reading a field, writing a field, and other Tag operations. 7. Statistics profiles to be included in the resulting command reports. 8. Mechanisms for validating the command cycle specifications.	3
3.5.2	Support for Command Reports 1. Description of how a command cycle was started and ended. 2. Description of what happened during the processing of a single Tag. 3. Information on the result of a single operation executing on a single tag during a command cycle. 4. The possible outcomes for a given operation **5.** Additional, implementation-defined information about each “sighting” of a Tag.	3
3.5.3	Writing tags 1. Allow upstream layers (e.g., BEG generator, applications) to formulate and populate the writing comments. To this end they may keep track of in-memory lists of tag values, along with patterns of tag values. 2. Support random number generators (RNG), as a source of random numbers that can be used by the write commands. 3. Deliver asynchronous results from command cycles to subscribers, through appropriate callbacks that deliver the command reports to subscribers through the notification URIs associated with them.	3

Table 10: Tag Writing Specifications

4.3.5 Logical Reader API

C/N	Specification	Priority
3.6.1	Provide a way for F&C clients to define a new logical reader name as an alias for one or more other logical reader names.	5
3.6.2	Manipulate “properties” (name/value pairs)	5

3.6.3	Access a list of all of the logical reader names that are available, as well as information about each logical reader.	5
3.6.4	Provide error handling capabilities.	5
3.6.5	Configure a way to reduce the appearance of tags moving in and out of a reader's field of view due to intermittent tag reads.	5

Table 11: Logical Reader Specifications

4.3.6 Access Control API

C/N	Specification	Priority
3.7.1	Access Control Mechanisms	3
3.7.2	Support for Access Control Identity	3
3.7.3	Support for Access Control Roles	3
3.7.4	Support for Access Control Permission	3
3.7.5	Support for anonymous user	3

Table 12: Specifications for Access Control to F&C Functionalities

4.3.7 ALE Management

C/N	Specification	Priority
3.7	**F&C Managemement**	4
3.7.1	Starting the F&C Server	4
3.7.2	Stopping the F&C Server	4
3.7.3	Managing Logical Readers	4
3.7.4	Managing Incoming Reports	4
3.7.5	Managing Outgoing Reports	4
3.7.6	Managing LLRP Specifications and Artifacts	3
3.7.7	Defined Reading Specifications (e.g., EPC-ALE ECSpecs)	3
3.7.8	Managing Subscribers	3

Table 13: F&C Management Specifications

5 Information Sharing Repository and Services Specification

5.1 Overview

The ASPIRE Information Sharing repository and services are the components that:

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

The Information Services of the ASPIRE Information Sharing middleware itself consists of three parts:

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

Note that the ASPIRE Information Sharing repository:

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

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

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

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

Please note that the ASPRIE Information Sharing Repositories Specification are influenced and compliant to the EPC Information Services (EPCIS) Specification Version 1.0.1[8].

5.2 Specification of Information Sharing Data Model

ASPIRE will exploit the data model defined in EPCIS. Hence, it will support the following events:

- Object Events, which correspond to observations of a collection of EPCs during a specific business step at a specified Location & Time.
- Aggregation Events, which reflect a physical association of a set of EPCs with a parent EPC along with a business step at a Location & Time.
- Quantity Events, which correspond to statements about an object Class (not individual objects), including a quantity, a Location & Time.
- Transaction Events, which records objects associated with a wider business transaction.

Similarly to EPCIS, ASPIRE Information Sharing services will be extensible. Extensibility will be supported in the following dimensions:

- **New Event Type:** This concerns the addition of a new Event Type.
- **New Event Field:** This concerns the addition of a new field to an existing Event Type.
- **New Vocabulary Type**: This relates to the addition of a new Vocabulary Type to the available Vocabulary Types.
- **New Master Data Attribute:** This relates to the definition of a new attribute name for an existing Vocabulary.
- **New Vocabulary Element:** This relates to the addition of a new element to an existing Vocabulary.

ASPIRE will capitalize on the notion of vocabularies in order to keep track of a company's (e.g., SME's) data. For the ASPIRE vocabularies a hierarchical or multi-hierarchical structure will be supported. Hierarchical relationships between vocabulary elements should be represented through master data. Specifically, a parent identifier carries, in addition to its master data attributes, a list of its children identifiers. Each child identifier must belong to the same Vocabulary as the parent.
In order to build vocabularies, the Information Sharing repository (and associated database schema) must support:

- Value Types Primitive types.
- Event Types.
- Event Fields included as part of the Event Types definitions.
- Vocabulary Types.
- Master Data Attributes included as part of Vocabulary Types definitions. In the scope of ASPIRE SMEs or even industry vertical working groups could define additional master data attributes for the vocabularies.
- Vocabulary Elements. It is expected that in the scope of ASPIRE SMEs or industry vertical working groups will define vocabulary elements for the BusinessStep vocabulary, the Disposition, and the BusinessTransactionType vocabulary.

Master data and event data will be hosted in a Relational Database according to the various events and vocabularies.

5.3 Information Sharing Services Specifications

As already outlined, information sharing services will undertake to access and write the Information Sharing data from/to the IS database. As already outlined Information Sharing services should provide two interfaces: (a) A capture interface fro writing to the repository and (b) A query interface for reading from the repository. The later must support both "pull" and "push" data access. The diagram below illustrates the relationship between these interfaces (capturing, query) as defined in the EPCIS specification [8].

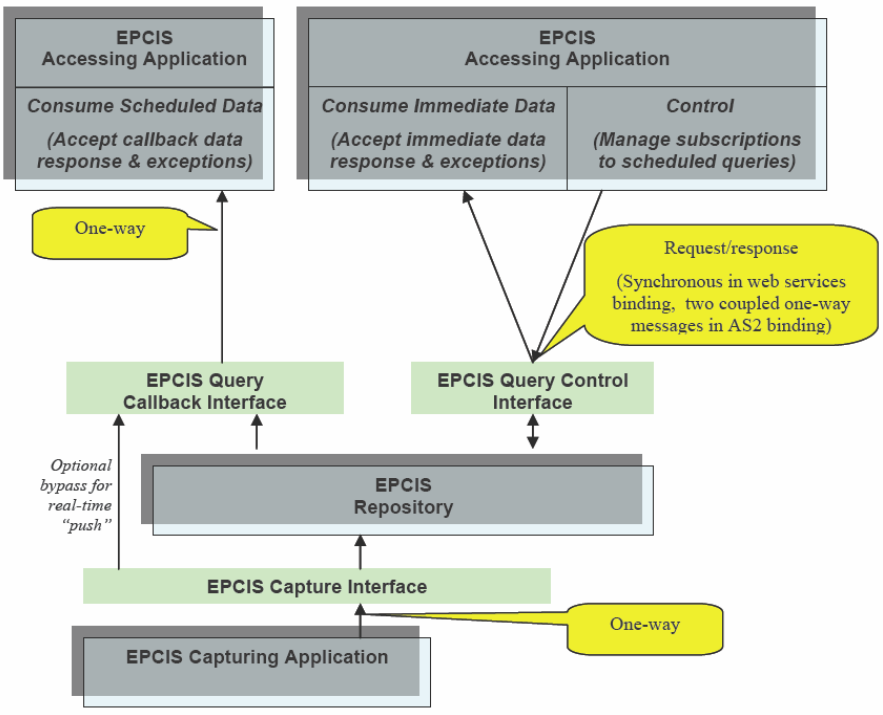

Figure 11 Query Control and Callback Interface relationship [8]

The ASPIRE Information Sharing specifications are in more detailed described in the following paragraphs.

5.3.1 Capture Operations

Capture Operations target the population of the Information Sharing respository based on core events, in the scope of a capture Application. In this context, we consider as "client" the capture application and "service" as a system that implements a capture interface.

5.3.1.1 Authentication and Authorization

At the capture interface there must exist a means for supporting the folliwng authentication operations:

- The "service" to authenticate the "client" identity,
- The client to authenticate the information sharing "Service's" identity
- Both of the above.

The means of authentication depends on the particular binding.

5.3.1.2 Event Capture

The capture interface must contain only a single method, capture, which takes a single argument and returns no results. The capture interface must accept each element of the argument list that is a valid Business Events (e.g., EPCISEvent) or subtype thereof. Other types of events through vendor or SMEs extension can also be accepted in the scope of the event capture operation.

5.3.1.3 Master Data Capture Service

The capture interface must contain only a single method, capture, which takes a single argument and returns no results. The capture interface must accept each element of the argument list that is a valid Master Data element.

5.3.2 Query Operations

Query operations must provide two interfaces:

- One for supporting business event data access on demand. Based on this interface upstream applications and other partners (e.g., trading partners) can capture data through interacting with the Information Sharing repository.
- One for supporting access to business event data based on subscriptions, through appropriate callbacks.

In this context the term “client” refers to application accessing the Information Sharing Repository, whereas the term “service” refers to a system that implements the above two interfaces (for “push” and “pull” data access).

5.3.2.1 Authentication

The ASPRIE Information Sharing middleware should provide a means for:

- The “service” to authenticate the “client” identity
- The “client” to authenticate the “Service” identity
- Both of the above operations.

5.3.2.2 Authorization

A “service” may wish to provide access to only a subset of information, depending on the identity of the requesting client. This situation commonly arises in cross-enterprise scenarios where the requesting client belongs to a different organization than the operator of a “service”, but may also arise in intra-enterprise scenarios.

5.3.2.3 Queries for Large Amounts of Data

Many of the query operations defined below allow a client to make a request for a potentially unlimited amount of data. For example, the response to a query that asks for all ObjectEvent instances within a given interval of time could conceivably return one, a thousand, a million, or a billion events depending on the time interval and how many events had been captured. This may present performance problems for service implementations.

To mitigate this problem, a “service” could reject any request by raising an exception. This exception indicates that the amount of data being requested is larger than the service is willing to provide to the client.

5.3.2.4 Overly Complex Queries

"Service" implementations may wish to restrict the kinds of queries that can be processed, to avoid processing queries that will consume more resources than the service is willing to expend. For example, a query that is looking for events having a specific value in a particular event field may require more or fewer resources to process depending on whether the implementation anticipated searching on that field (e.g., depending on whether or not a database column corresponding to that field is indexed). As with queries for too much data, this may present performance problems for service implementations.

To mitigate this problem, a "service" MAY reject any request by raising an exception. This exception indicates that structure of the query is such that the service is unwilling to carry it out for the client.

5.3.2.5 Query Framework

The ASPIRE Information Sharing Query mechanism will provide a general framework by which client applications may query business events. The mechanism should provide both on-demand queries, in which an explicit request from a client causes a query to be executed and results returned in response, and standing queries, in which a client registers ongoing interest in a query and thereafter receives periodic delivery of results via the a callback mechanism.

5.3.2.6 Error Conditions

Methods of the Query Control API should signal error conditions to the client by means of exceptions. In addition to exceptions thrown from the Query Control Interface, an attempt to execute a standing query may give an exception (e.g., in case the results returned by the query are too large).

5.3.2.7 Predefined Queries for Information Sharing

The ASPIRE Information Sharing implementation should provide predefined queries, which a client may invoke using the poll and subscribe methods of the Query Control Interface / API. The predefined queries defined in this section each have a large number of optional parameters; by appropriate choice of parameters a client can achieve a variety of effects.

5.3.2.7.1 Simple Event Query

This query is invoked by specifying the quary as a string, which the implementation has to either poll or subscribe. The result should contain a (possibly empty) list of Event instances. Each element of the result list could be of any event type; i.e., ObjectEvent, AggregationEvent, QuantityEvent, TransactionEvent, or any extension event type that is defined as part of the extension mechanisms.

5.3.2.7.2 Simple Master Data Query

This query is invoked by specifying a query as a string to the polling function. The result should contain a (possibly empty) list of vocabulary elements together with selected attributes. The SimpleMasterDataQuery should be available via poll but not via subscribe; an exception should an exception in case subscription is accepted.

5.3.2.8 Query Callback Interface

The Query Callback Interface is the path by which a "service" delivers standing query results to a client. A callback method should be called each time the "service" executes a standing query.

5.3.3 Bindings for Capture and Query Operations

ASPIRE will support messaging binding for capture operations. In particular, message queues, based on both point-to-point and publish/subscribe XML messaging. The implementation of message queue should be based on JMS technology. In addition HTTP bindings for the capture operations will be supported.

Likewise for the Query Control Interface SOAP/HTTP, AS2 Binding, HTTP Binding and HTTPS bindings should be supported.

5.3.4 Management of Information Sharing Repository and Processes

The repository will be implemented as a JMX-enabled management application, to allow for flexible management of the Information Sharing servers and repository. As shown in Figure 14 of the JMX Architecture three layers should be implemented:

- Instrumentation Level.
- Agent Level and
- Adaptors level

For every resource that needs management and monitoring the instrumentation process will be implemented. A Java objects known as MBeans following the design patterns and interfaces defined in the JMX specification will be used for each and one of them to expose the management information in the form of attributes and operations and offer access to the instrumentation of resources. MBeans for the following functions MAY be created:

- Starting the various components of the repository.
  This will mainly be achieved by bundling the various components to work within an OSGI container (according to the ASPIRE architecture).
- Stopping the various components of the information sharing repository.
  This will mainly be achieved by bundling the various components to work within an OSGI container (according to the ASPIRE architecture).
- Managing the Captured data.
- Managing the Queried data.

.Also an MBeanServer will be created which will contain the list of MBeans registered with it. All management operations performed on the MBeans will be done through the MBeanServer. All the JMX agents that will provide the set of services will reside at the MBeanServer. Each of these services is termed an agent service.

The JMX agent should contain at least one protocol adaptor or connector. These protocol adaptors and connectors provide the possibilities of remote management, by defining the manager components which are capable of communicating with the agents. Protocol adaptors and connectors make the

agent accessible from remote management applications. They provide a view through a specific protocol of the MBeans instantiated and registered in the MBean server. For exporting JMX API instrumentation to remote applications Remote Method Invocation (RMI) will be used.

5.4 Information Sharing Specifications Overview

The following table summarizes the ASPIRE Information Sharing Specifications. Implementation priority values from 1 to 5 with 5 being of most importance.

C/N	Specification requirements	Priority
5.1	**Information Sharing Data Model**	4
5.1.1	Extension Mechanisms	4
5.1.2	Hierarchical Vocabularies	4
5.1.3	Support for Core Event Types (EPCIS compliant)	4
5.2	**Capture Operations**	4
5.2.1	Authentication and Authorization	3
5.2.2	Event Capture Service	5
5.2.3	Master Data Capture Service	5
5.3	**Query Operations**	4
5.3.1	Authentication	3
5.3.2	Authorization	3
5.3.3	Queries for Large Amounts of Data	3
5.3.4	Overly Complex Queries	3
5.3.5	Query Control API	4
5.3.6	Error Handling	3
5.3.7	Simple Event Query	5
5.3.8	Simple Master Data Query	5
5.3.9	Query Callback Interface	5
5.5	**Bindings for Capture Operations**	2
5.5.1	Message Queues	2
5.5.2	HTTP	2
5.6	**Bindings for Query Operations**	4
5.6.1	Query Control Interface	4
5.6.1	SOAP/HTTP	2
5.6.2	AS2 Binding for the Query Control Interface	2
5.6.2	Query Callback Interface	2
5.6.2.1	HTTP Binding	2
5.6.2.2	HTTPS Binding	2
5.6.2.3	AS2 Binding	2
5.7	**Management Operations**	3
5.7.1	Starting	3
5.7.2	Stopping	3
5.7.3	Captured data	3
5.7.4	Queried data	3

Table 14: Overview of Specifications for the ASPIRE Information sharing repository

6 Business Event Generation Specifications

6.1 Overview

The F&C (Filtering and Collection) server sends specific reports about the collected data to external applications. Such data are collected and read according to reading specifications that are described above and which are set on the F&C middleware. The F&C middleware will in principle output reports that contain raw RFID tag streams (e.g., tag IDs). The latter do not carry any business semantics and cannot therefore be understood and stored in the database structure of the information sharing repository. Hence, there is a clear need for an application that will add business semantics to the raw tag streams, thus enabling their storage and manipulation from the ASPIRE information sharing middleware. This application for example can put the "sighting" of a tag to a particular business context comprising the company, the business location, the read point, as well as the business process in the scope of which the particular tag was read by the F&C layer.

Adding appropriate business context to raw tag streams requires access to the master/company data of the information sharing repository, as well as to run-time parameters that instantiate the business context generation. Thus, RFID deployments have to rely in capturing applications that take into account the particular business semantics pertaining to the target business case. ASPIRE automates this process based on the specification of a generic and configurable middleware component for Business Event Generation (BEG). The BEG will incorporate middleware logic for:

- Looking up master data at the ASPIRE information sharing repository.
- Extracting the required business semantics (metadata).
- "Decorating" tag information with business semantics in order to generate business events that comply to the ASPIRE information sharing specifications.

The BEG will of course leverage run-time parameters for identifying and customizing its operation to particular instances of business processes and tags. These run-time parameters will for example tailor the BEG engine to work with a particular invoice or pro-form (hence creating a BEG instance for the particular task at hand). As shown in Figure 12 the BEG will query the Information Sharing Master Data vocabularies so as to collect the information needed, that will already have been inserted to the Master Data repository, to get the necessary context for interpreting the incoming data reports generated from the F&C server.

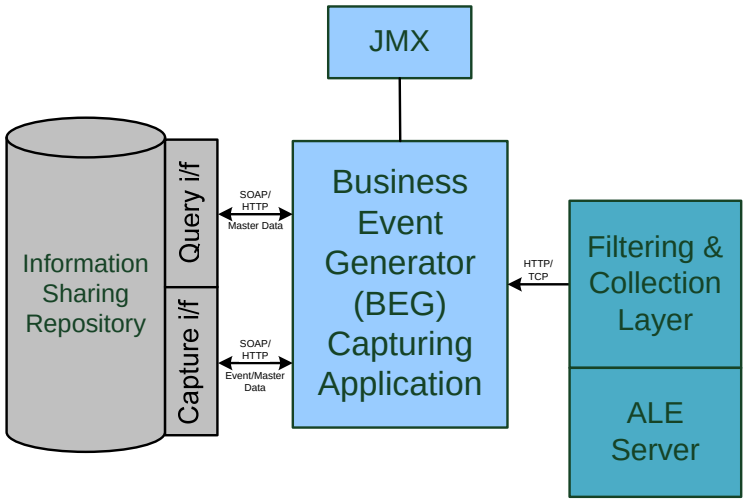

Figure 12 The Business Event Generation (BEG) stands between the F&C and Information Sharing Layers

6.2 BEG Engine Specification

6.2.1 BEG to F&C bindings

ASPIRE BEG will implement HTTP/TCP to receive asynchronous results through a direct callback. Hence, the BEG shall be working in an asynchronous fashion (i.e. waiting from tag streams from the F&C).

6.2.2 BEG to Information Sharing bindings

The BEG implementation will provide a SOAP/HTTP binding for interfacing with the middleware of the information sharing layer and the associated repository.

6.2.3 Access/Collect required Master Data

BEG should query the repository in order to collect the required attributes from the Business Transaction Vocabulary of predefined Events. Hence, BEG should first associate a tag with a particular event (e.g., an Object Event if the object is an item, an Aggregation event if the object contains other objects (i.e. it is a palette or container or package or shelf).

6.2.4 Reports Processing

BEG should be able to use the predefined business transactions attributes that it will acquire and should accordingly use them to “decode” and “decorate” the delivered report from the F&C server.

6.2.5 Authentication and Authorization

The BEG engine must be seen as a capture application i.e. a “client” application for the information sharing repository. Hence, BEG access should be able to authenticate itself against the Information Sharing capture operations authentication layer.

6.2.6 BEG Management

BEG will be also JMX-enabled management application. As shown in Figure 14 of the JMX Architecture three layers should be implemented:

- Instrumentation Level

- Agent Level and
- Adaptors level

For every resource that needs management and monitoring the instrumentation process will be implemented. A Java objects known as MBeans following the design patterns and interfaces defined in the JMX specification will be used for each and one of them to expose the management information in the form of attributes and operations and offer access to the instrumentation of resources. MBeans for the following functions may be created:

- Starting the various components of the BEG application.
  This will be achieved by bundling the various components to work within an OSGI container.
- Stopping the various components of the BEG application.
  This will mainly be achieved by bundling the various components to work within an OSGI container.
- Managing the Incoming Reports (i.e. tag streams from the underlying F&C server).
- Managing the selected transaction to capture.

Also an MBeanServer will be created which will contain the list of MBeans registered with it. All management operations performed on the MBeans will be done through the MBeanServer. All the JMX agents that will provide the set of services will reside at the MBeanServer. Each of these services is termed an agent service.

The JMX agent should contain at least one protocol adaptor or connector. These protocol adaptors and connectors provide the possibilities of remote management, by defining the manager components which are capable of communicating with the agents. Protocol adaptors and connectors make the agent accessible from remote management applications. They provide a view through a specific protocol of the MBeans instantiated and registered in the MBean server. For exporting JMX API instrumentation to remote applications Remote Method Invocation (RMI) will be used.

6.2.7 Graphical User Interface

The BEG engine (capturing application) should come with a simple GUI that will provide the following functionalities:

- Entering the port that the BEG will listen for the reports from the F&C server.
- Entering the information sharing server URI that will provide the Capture/Query Interface service.
- Choosing one from predefined business transactions for the report processing and instantiating with needed parameters (e.g., the transaction identification).

This GUI will be incorporated to the overall ASPIRE IDE. Note however that the parameters specified above (i.e. port, URI, business transaction identification) specify a BEG engine instance. Hence, the ASPIRE IDE should also allow for abstracting the above information and associating it with an alias for flexible management of BEG engine instances. This will allow using the alias for development/deployment, rather than carrying the low-level information.

6.3 BEG Specifications Overview

The following table summarizes the ASPIRE BEG specifications. Implementation priority values from 1 to 5 with 5 being of most importance.

C/N	Specification	Priority
4.1	**BEG to F&C bindings**	5
4.2	**BEG to information sharing bindings**	5
4.3	**Master Data Access and Collection**	5
4.4	**Reports Processing**	5
4.5	**Authentication and Authorization**	3
4.6	**BEG Management**	4
4.6.1	Starting	3
4.6.2	Stoping	3
4.6.3	Incoming Reports	3
4.6.4	Captured Transaction	3
4.7	**Graphical User Interface**	4

Table 15: BEG Specifications requirement Overview

7 Connector Specifications

7.1 Overview

RFID middleware components described in the previous paragraphs provide a foundation for translating raw RFID streams to meaningful business events comprising business context such as where a tag was seen, at what time and in the scope of which process. Enterprises can then leverage these business events through their legacy IT systems (e.g., ERPs, WMS, corporate databases), which are used to support their business processes. To this end, there is a clear need for interfacing these legacy systems, with the information sharing repositories, established and populated as part of the RFID deployment. Interfacing between IT systems and the information sharing repository, as well as other middleware blocks of the RFID deployment is realized through specialized middleware components that are called “connectors”.

The main purpose of connector components are to abstract the interface between the ASPIRE information repository and enterprise information systems. Hence, connectors offer APIs, that enable proprietary enterprise information systems to exchange business information with the an ASPIRE RFID middleware system.

Connectors should therefore provide:

- **Support for services and events:** Composite applications should be able to call out to existing functionality as a set of services, and to be notified when a particular event type (for example, “purchase order inserted,” “employee hired”) occurs within an existing application. Typical events of interest to the enterprise information system are those signifying the boundaries of a transaction (i.e. transaction start and transaction finish events).
- **Service abstraction:** All services should have some common properties, including error handling, syntax, and calling mechanisms. They should also have common access mechanisms such as JCA (Java Connector Architecture), JDBC, ODBC (Object Database Connectivity), and Web services, ideally spanning different platforms. This makes the services more reusable, while also allowing them to share communications, load balancing, and other non-service-specific capabilities.
- **Functionality abstraction:** Individual services must be driven by metadata about the transactions that the business needs to execute. Ideally, this metadata is stored in a platform-agnostic and easily transformed format so that the interfaces can be easily adapted to new technologies.
- **Process management:** Services should embed processes, and process management tools should call services. Hence, connectors should be able to support the grouping of several service invocations to processes..

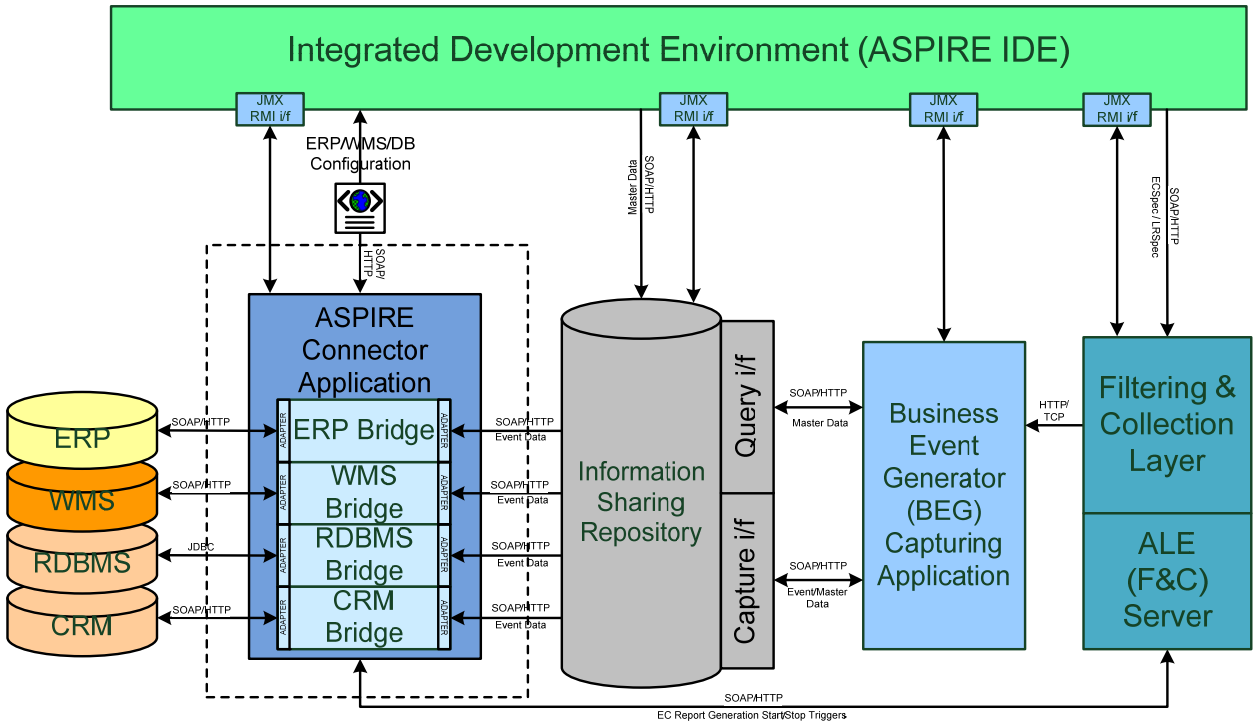

Figure 13 Overview of a Connector Application and its positioning in the ASPIRE architecture

7.2 Connector Specifications

7.2.1 Adapter Framework

7.2.1.1 Standard Adapters for Information Exchange – Information Exchange Semantics

In the scope of the interaction between enterprise information systems and the ASPIRE information sharing repository, the ASPIRE connector application may support popular adapter frameworks [14] for information exchange, such as ebXML (Electronic Business using eXtensible Markup Language) and EDI (Electronic Data Interchange). Depending on the application, ASPIRE may also consider connectors handling specialized semantics e.g., HIPAA (Health Insurance Portability and Accountability Act) for healthcare applications and ISO 15022 by for financial services.

7.2.1.2 Standard interfaces – Application and Data Adapters

The connector application has to support standard popular enterprise interfaces for interacting with IT systems such as:

- Java Database Connectivity (JDBC) for interacting with relational databases.
- The JavaEE™ Connector Architecture (JCA) for interacting with JCA enabled applications.
- W3C Web Services for interacting with numerous applications that provide nowadays Web Services interfaces.

The Connector application should use standard APIs and document interfaces to integrate applications at the application logic level. It will reuse the business

rules, error handling, and processing logic already built into the application. This reduces the risk of data integrity violations and avoids adapter recreation when the underlying database changes.

7.2.1.3 Custom tooling for application platform suites

The Connector application should come with custom tooling for integration within the ASPIRE IDE and possible other (third-party) tools.

7.2.1.4 Transaction processing adapters

Connector application should provide transaction processing support through leveraging the native transaction processing capabilities of the underlying business systems. The objective will be to reuse the business logic that they encapsulate while ensuring transactional integrity.

7.2.2 Graphical User Interface

The Connector application shall provide a UI for providing feedback to the user for its various functionalities and configuration. The UI should display information related to individual tagged objects as well as logical or physical groups of tagged objects. The information should be displayed with rich widgets according to the nature or type. For instance, if the history of a tagged object is associated with GPS positions, the track of the object should be displayed in geomap-based widgets such as Google Maps or Yahoo Maps. In the same way, the history of a tagged object is associated with temperature values, the temperature curve should be shown to the user. The curve should include also flags for threshold infringements.

7.2.3 Connector to Various Systems ERPs bindings

Connector Application should use service adapters to create an enterprise SOA (Service Oriented Architecture) with the various ERP, WMS and SCM systems. It will have a collection of standards-based interfaces to business functions.

Specifically it should provide a SOAP/HTTP binding to connect with the various applications (ERP, CRM, WMS etc.); if a SOAP/HTTP binding is provided, it should conform to the WSDL. This SOAP/HTTP binding is compliant with the WS-I Basic Profile Version 1.0 [20]. This binding builds upon the schema.

If a connector providing the SOAP binding receives an input that is syntactically invalid according to this WSDL, the implementation should indicate this in one of the two following ways:

- the implementation may raise a ValidationException, or
- it may raise a more generic exception provided by the SOAP processor being used.

7.2.4 Connector to Various RDBMS

As already outlined the connector application should implement JDBC for interfacing with various RDBMS systems.

7.2.5 Connector to ASPIRE Information Sharing repository and services

The connector implementation should provide a SOAP/HTTP binding to interface with the ASPIRE Information Sharing repository.

7.2.6 Connector to F&C bindings

Connector implementation should provide a SOAP/HTTP binding to interface with the F&C server.

7.2.7 Authentication and Authorization

The connector application should provide a means to authenticate the client's identity. Hence, the connector should be authenticated against the Information Sharing authentication mechanism (established at the capture interface and the various applications).

7.2.8 Connector Management

A connector will be a JMX-enabled management application. As shown in Figure 14 of the JMX Architecture three layers should be implemented:

- Instrumentation Level
- Agent Level and
- Adaptors level

For every resource that needs management and monitoring the instrumentation process will be implemented. A Java objects known as MBeans following the design patterns and interfaces defined in the JMX specification will be used for each and one of them to expose the management information in the form of attributes and operations and offer access to the instrumentation of resources. MBeans for the following functions may be created:

- Starting the various components of the Connector.
  This will mainly be achieved by bundlizing the various components to work within an OSGI container.
- Stopping the various components of the Connector.
  This will mainly be achieved by bundlizing the various components to work within an OSGI container.
- Managing the various implemented adapters
- Managing the connections between the systems

Also an MBeanServer will be created which will contain the list of MBeans registered with it. All management operations performed on the MBeans will be done through the MBeanServer. All the JMX agents that will provide the set of services will reside at the MBeanServer. Each of these services is termed an agent service.

The JMX agent should contain at least one protocol adaptor or connector. These protocol adaptors and connectors provide the possibilities of remote management, by defining the manager components which are capable of communicating with the agents. Protocol adaptors and connectors make the agent accessible from remote management applications. They provide a view through a specific protocol of the MBeans instantiated and registered in the MBean server. For exporting JMX API instrumentation to remote applications Remote Method Invocation (RMI) will be used.

7.3 Connector Specifications Overview

The following table summarizes the ASPIRE Connector specifications. Implementation priority values from 1 to 5 with 5 being of most importance. Note that the starting point for the ASPIRE connector applications is its interface to corporate databases (i.e. the implementation of data adaptors and connections to popular RDBMS systems). These tasks feature the highest priority in the table.

C/N	Specification Requirements	Priority
6.1	**Adapter Framework**	
6.1.1	Standard Adapters (eBXML, EDI)	1
6.1.2	Standard interfaces (Web Services)	1
6.1.3	Custom tooling for application platform suites	1
6.1.4	Application adapters	4
6.1.5	Data adapters (JDBC)	5
6.1.6	Transaction processing adapters	1
6.2	**Graphical User Interface**	3
6.3	**Connector to Various Systems ERPs bindings**	4
6.4	**Connector to Various RDBMS (JDBS)**	5
6.5	**Connector to EPCIS bindings**	5
6.6	**Connector to F&C bindings**	3
6.7	**Authentication and Authorization**	2
6.8	**Connector Management**	3
6.8.1	Starting	3
6.8.2	Stoping	3
6.8.3	Adapters	3
6.8.4	Connections	3

Table 16: Connector Specifications Overview

8 ASPIRE IDE and Tools Specifications

8.1 Overview

ASPIRE will implement a number of editing and management tools enabling RFID consultants and/or users to easily build and deploy RFID solutions. The purpose of these tools will be twofold:

- To minimize the programming and configuration effort required to implement and fully leverage an RFID solution.
- To manifest the programmability capabilities of the ASPIRE middleware platform, through demonstrating that end-to-end RFID solution can be essentially built and deployed using the ASPIRE tools.

The ASPIRE editing tools will deal with specification and configuration of middleware functionalities. The tools will be integrated in a single integrated development environment (IDE) for RFID applications, which we conveniently call ASPIRE IDE.

8.2 Management Console Specifications

The ASPIRE IDE management console will use the JMX abstraction layer to request from the managed objects to execute specific tasks and it will use Remote Method Invocation (RMI) handles the communication between manager and JMX agent.

The abstraction layer will utilize JMX adapters to communicate with the actual managed objects. JMX adapters are the components that incorporate the object specific management logic and provide a standardized interface to the abstraction layer.

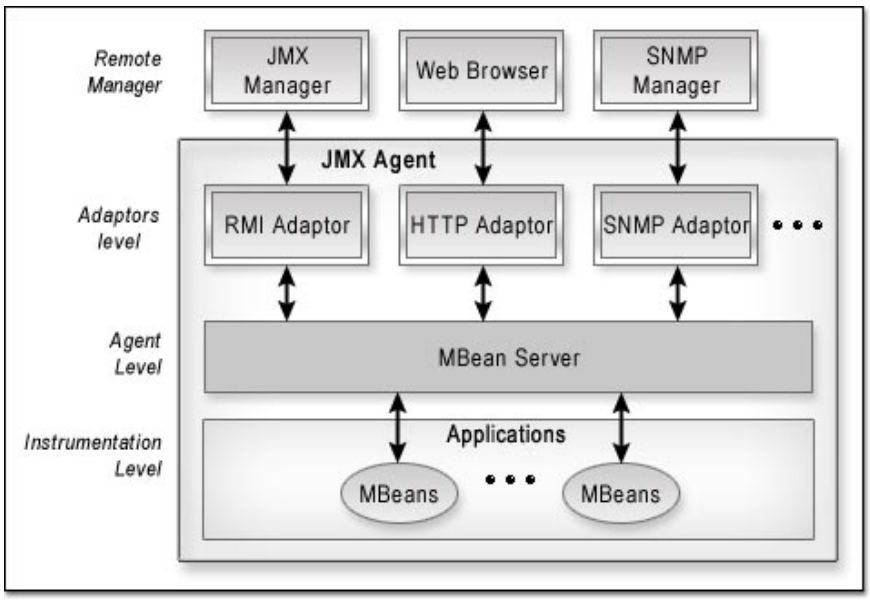

Figure 14 Overview of JMX Architecture [14]

The Aspire management component will require a JMX adapter interface for every manageable component in order to function properly and be able to fulfill its goal.

JMX technology is defined by two closely related specifications developed through the Java Community Process (JCP) as Java Specification Request (JSR) 3 and JSR 160:

- JSR 3, Java Management Extensions Instrumentation and Agent Specification and
- JSR 160, Java Management Extensions Remote API.

For the RCP console
The management architecture can be broken down into three levels. The first two levels shown below, instrumentation and agent are defined by JSR 3. The remote management level is defined by JSR 160.

- **Remote Management:** Protocol adaptors and standard connectors make a JMX agent accessible from remote management applications outside the agent's Java Virtual Machine (JVM).

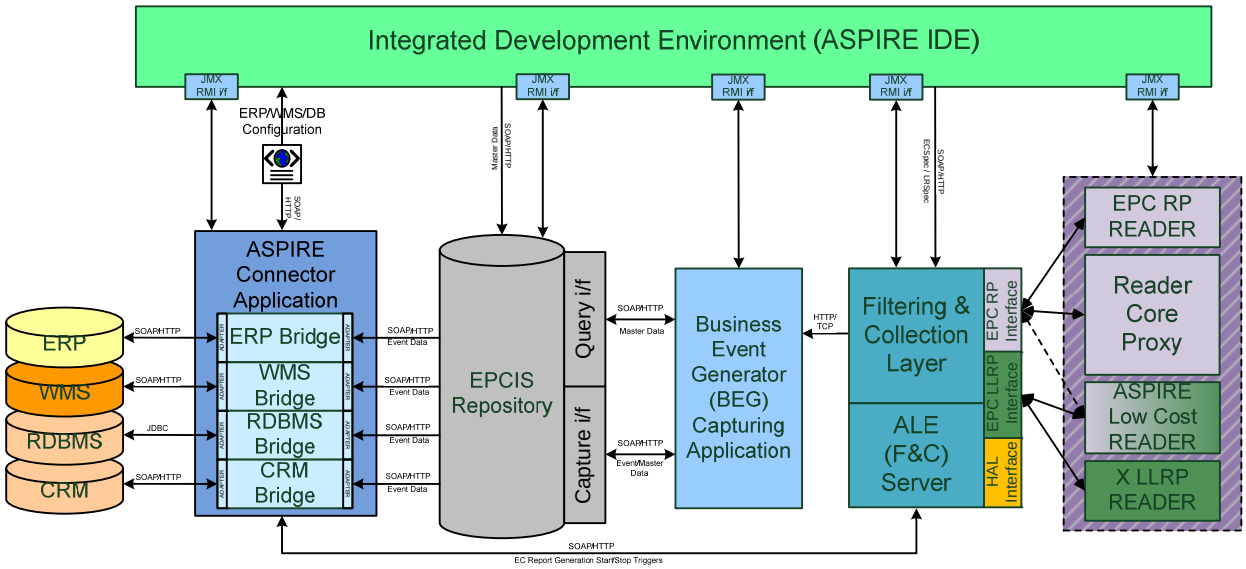

Figure 15 ASPIRE IDE management Architecture

The ASPIRE IDE management console should manage as shown in Figure 15 the following components:

- The various RFID readers that support RM (Reader Management)
- Reader Core proxy
- F&C Server
- BEG engine
- EPCIS repository
- Connector application

8.3 Tooling Specifications

The ASPIRE IDE components SHOULD provide means of configuration of the underlying ASPIRE infrastructure. The user by describing his requirements to the IDE, which should provide all the configuration options, will "translate" them into configuration messages by which it will supply all the appropriate modules.

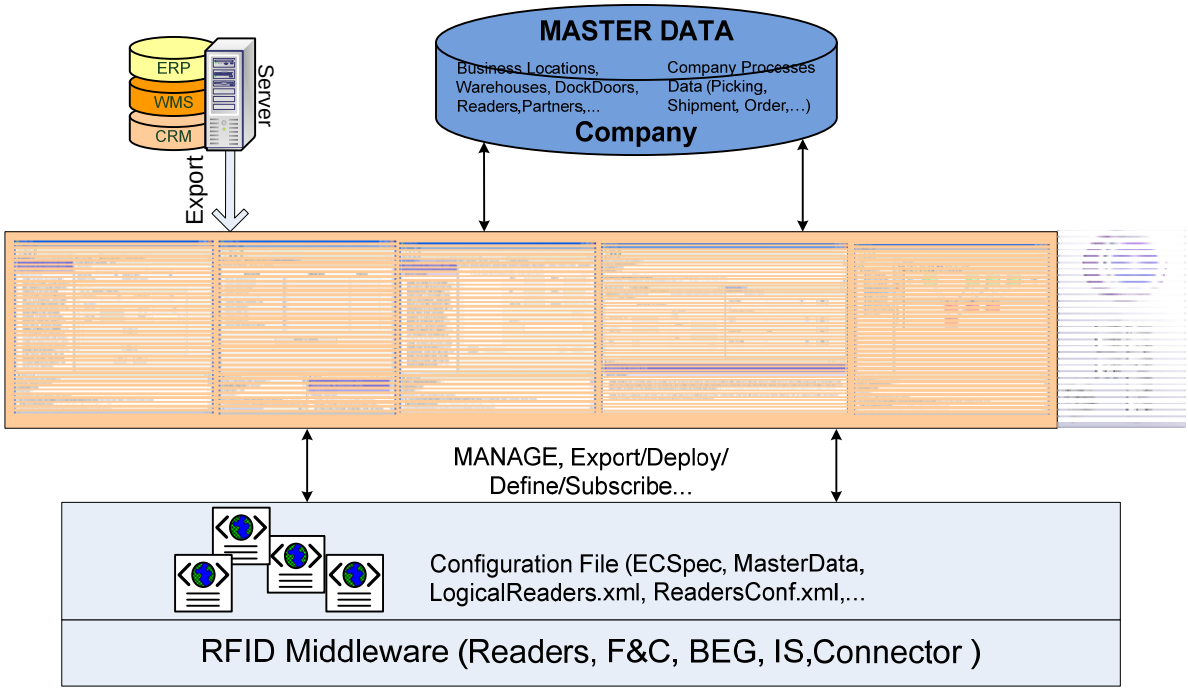

Figure 16 ASPIRE IDE Tools

8.3.1 ASPIRE IDE

ASPIRE IDE has been designed as an Eclipse RCP (Rich Client Platform) application that will run over Equinox OSGI server. Every tool should be an eclipse plugin/bundle that will be able to be installed or removed as needed. This way many editions of the ASPIRE IDE can be released depending on the functionalities required (as simple or as complicate depending on the demands) for the RFID middleware that will be implemented.

8.3.2 Physical Reader Configuration Editor

This is a tool enabling consultants to configure physical readers and their operational parameters and environments. This tool should be seen as complementary to vendor specific tools (i.e. tools that come with each of the readers). It should support the basic functionalities for an RFID reader (e.g. field strength, number of antennas used, antenna tries, read tries, write tries, e.t.c.)

8.3.3 Logical Reader Configuration Editor

This tool will support the definition of logical readers (e.g., based on clustering of physical, simulation and proxy readers). Four kinds of readers will be supported:

- LLRP readers
- RP readers
- HAL readers and

- Simulator readers

8.3.4 Reading Specifications Editor

This is a tool enabling editing, as well as management of F&C server Filtering Specifications. It will be able to create, load and edit a reading specifications XML file (e.g., an ECSpec according to the EPC Global ALE 1.1. specification).

8.3.5 F&C Commands Execution

The objective of this tool is to provide a control client to execute Application Level Event specification (ALE) commands on a reader or component that implements the ALE specification.

8.3.6 Connector Configurator

This tool will be able to interact with the Connector application to reveal all its functionalities and configurations. It should enable configuration of connectors to different systems and databases.

8.3.7 Master Data Editor (with support for Elementary Business Process Description)

A master data editor will be provided, enabling users and/or consultants to edit enterprise data including information about the company's location, its business locations, readpoints, as well as its business processes. The description of information such as business locations and readpoints is straightforward. The description of business processes is indeed more challenging, since it requires mapping business requirements to collections of RFID business events (according to the ASPIRE information sharing framework established in earlier sections). The starting point is the documentation of the business requirements, comprising the archetypical use cases. It is possibly to use the master data concept in order to encode business locations, read points, logical warehouses, containers, disposition states, as well as event and business steps sequences as shown in Figure 16 ASPIRE IDE Tools. In order to provide a more general framework for handling RFID enabled logistics possesses we suggest encoding the above data within a processes description language that could be amenable by graphical tools.

It is important to break each use case into a series of discrete business steps corresponding to various business events. Fixed lists of identifiers with standardized meanings for concepts like business step and disposition must be defined, along with rules for population of user-created identifiers like read point, business location, business transaction and business transaction type. All these information elements will be stored and managed as pieces of Master Data, within an appropriate database schema.

An example in the area of warehouse management is given in Appendix A, where popular processes [21] e.g. receiving, moving within warehouses, order collection, pick & pack, order shipment, inventory are described in terms of RFID business events. Figure 17 depicts the concept of decomposing a process into a number of business events. The later events comply to the ASPIRE Information Sharing specifications for RFID events (with direct references to EPC-IS framework). We call Elementary Business Process, the process which can be

directly decomposed into RFID business events (as shown in Figure 17). As already noted, Appendix A, includes the description of a number of typical elementary business processes in the area of warehouse management and logistics [16].

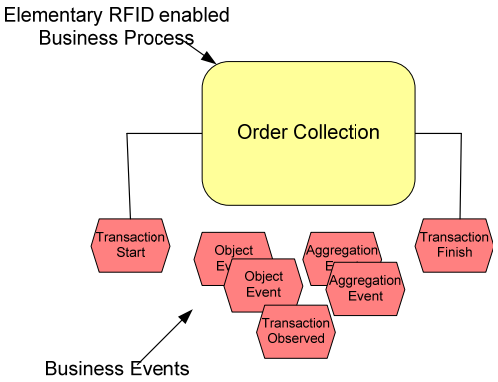

Figure 17: Description of Elementary RFID enabled Business Process

Summing up the Master Data Editor will enable editing of a company's metadata (Master Data), located at the Information Sharing repository needed to describe the company's operations, which includes:

- **Business dispositions** where one can set:
  - its id
  - its name
  - and attributes as needed
- **Business steps** where one can set:
  - its id
  - its name
  - and attributes as needed
- **Business transactions** where one can set:
  - The transaction id
  - The transaction name
  - Attributes as needed
  - Events that belongs to this transaction
    - Event business location
    - Event type
    - Business step for the event
    - Disposition for the event
    - Read point
    - Event id
    - Report name this event is bind with
- **Transactions type** where one can set:
  - its id.
  - its name.
  - and attributes as needed.
- **Business locations** where one can set the company's:
  - Name.
  - Address.
  - Country.
  - City.

  - Read point.
  - And a description.
- Finally **Read Points** where one can set
  - Its id.
  - Its name.
  - And attributes as needed.

Note that some of the above master data should be edited in an hierarchical fashion, i.e. Business Locations can in principle have as children other Business Locations, while transactions could in principle have as children other Business Transactions. This concept is also illustrated through an example in Appendix A, where a typical taxonomy of a company's warehouse is illustrated. This typical taxonomy reveals that a company's warehouse spaces (e.g., shelves, rooms, warehouse, central warehouses) are indeed organized in an hierarchical fashion, which must be supported by the ASPIRE tooling.

8.3.8 ASPIRE Business Process Management and Workflow Management Editor for Composite Business Processes

In the scope of the previous paragraph we define the notion of an elementary RFID enabled business process. Implementing elementary business processes (e.g., those illustrated in Appendix A) is certainly the first step for companies that would to bootstrap RFID deployment. We envisage however that the implementation of more than one process, along with their integration (e.g., connecting the pick & pack with the shipment process in the case of the processes exemplified in Appendix A) shown in Figure 18 can enable higher degrees of automation and efficiency, overall yielding a multiplicative benefit. Hence, moving at a higher business processes management level, it is possible to define how individual business transactions and processes are connected and/or integrated in the scope of a wider process.

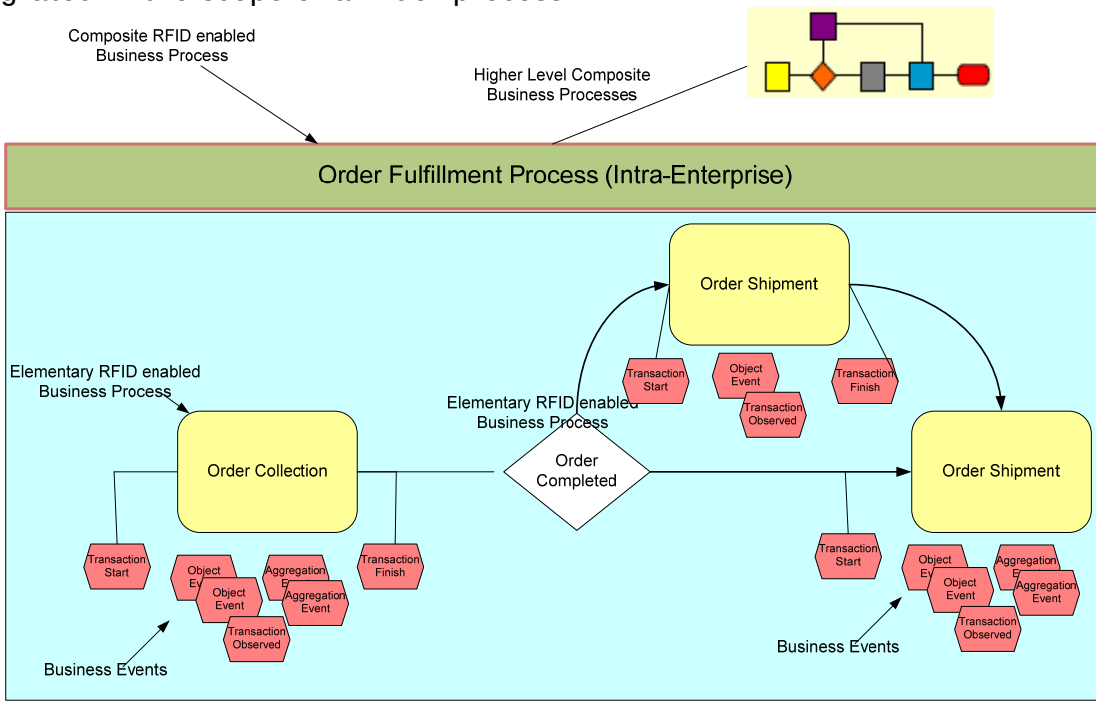

Figure 18: Connecting Elementary RFID Business Processes to Composite Business Processes- Process Management

The ASPIRE tools will include data workflow management editor, which can offer to business users and RFID consultants a graphical user interface for manipulating complex business processes, that are composed as workflows of elementary business processes as shown in Figure 18. The workflow management tool will leverage (though. using/ invoking) the functionality of the above tools in order to support RFID deployments in accordance to particular business processes. The workflow management tool will use a graphical editor (e.g., based on the XPDL (XML Process Definition Language)) to enable business users to describe and configure all of the company's assets/business processes (Master Data), as well as functional specifications with the help of a workflow business diagram.

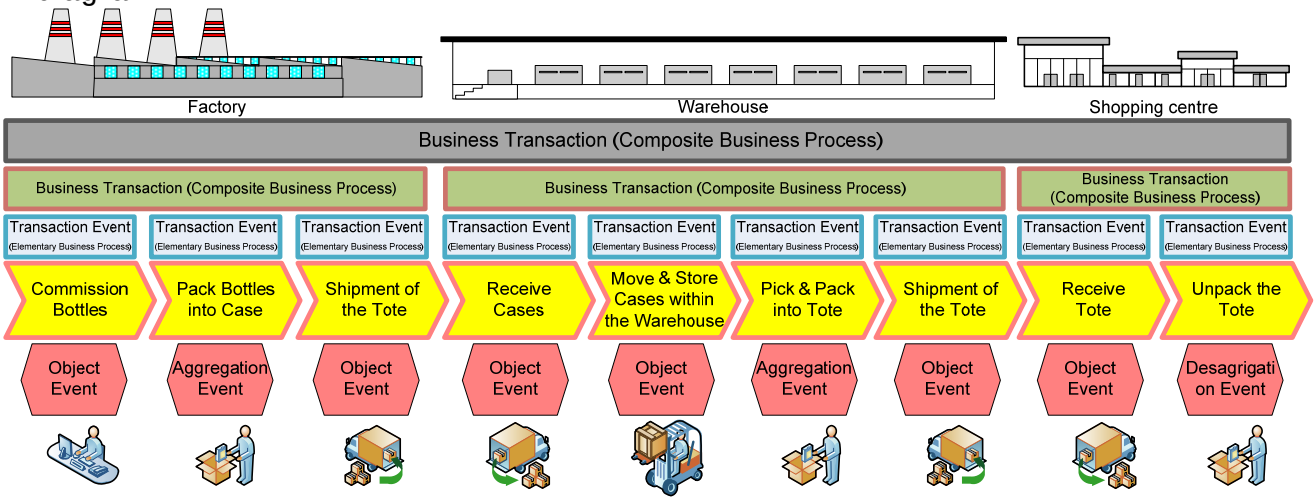

Figure 19: Wider Business Process/Transactions Example

8.3.9 ASPIRE Programmability Engine

Although not a distinct tool itself, the ASPIRE IDE will include a programmability engine, which should be an integral component of the ASPIRE IDE. This engine will be able to process a fully fledged RFID solution described in a special purpose domain specific language (e.g., an XML based language). This language will be specified as part of future deliverables of the WP4 of the ASPIRE project. The concept of a Domain Specific Language for RFID is illustrated in [18].

The ASPIRE solution description language will comprise metadata associated with the above tools (e.g., the master editor's metadata, process language output of the workflow editor (e.g., XPDL files)) for a specific RFID solution. Hence, the solution description language will include a complete description of an RFID solution. Along with this language the project should implement run-time middleware engines, which will enable the translation of the solution language to the various specification files (Reading specifications (e.g., ECSpecs), Master Data, Connector Specifications, Logical Readers configuration files, LRSpec for the LLRP protocol), which are required to deploy a specific solution over the ASPIRE RFID middleware platform. Moreover, the engine will be able to carry out the reverse process i.e. get as input the various configuration files (Reading

specifications (e.g., EPC-ALE ECSpecs), Master Data, Connector Specifications, Logical Readers configuration files, LRSpec for the LLRP protocol) and use them to build the ASPIRE specific solution language (e.g., an XML or XPDL based description). In this way business users and/or consultants will be able to use the workflow graphical editor to edit processes and accordingly to use them for reconstructing the specifications required to deploy (the updated) solution to the underlying ASPIRE middleware. The concept of specifications' generation is illustrated in Figure 16, which assumes that all the ASPIRE tools are unified and integrated in an integrated development environment namely the IDE.

8.3.10 ASPIRE Tools Summary

The following table (Table 17) depicts the classification of F&C functionalities in the above areas, also outlining some characteristic use cases where they are needed. It is evident that the specified F&C layer addresses several key requirements and use cases of Automatic identification applications.

ASPIRE Tool	Key Functionalities	Sample Use Cases
Physical Reader Configuration Editor	Tool enabling access to low-level functionalities of a physical reader	• Configure a reader's power and/or read range • Mange reader information and configuration parameters
Logical Reader Configuration Editor	Tool enabling definition of logical reader configurations on the basis on various underlying physical readers and antennas	• Build a logical reader configuration (for a dock-door portal) comprising X1 readers and Y1 antennas. • Amend the above configuration to include X2 readers (instead of X1) Y2 antennas (instead of Y1)_
Reading Specifications Editor	Tool enabling creation/editing and deletion of reading specifications.	• Configure filters to read only tags of interest to the particular application/deployment. • Configure particular tag groupings to be reported in the report.
F&C Commands Execution	Tool enabling execution of commands regarding reading specifications on the F&C server	• Establishing a reading specification on the F&C server • Abolishing a reading specification on the F&C server.
Connector Configurator	Configures connections from the ASPIRE middleware to legacy IT systems and databases	• Define the connection to a database and map transactions to tables.
Master Data Editor	Edit company data including business	• Define a business processes as a number of distinct

	locations, readpoints and business processes.	business steps each one corresponding to an event. • Define an hierarchical tree of business locations.
Business Process Management / Workflow Management Tool	Allow companies (SMEs) to compose complex RFID business processes, as workflows of elementary business processes.	• Create, edit, delete processes flows comprising RFID enabled operations. • Create hierarchical business processes.

Table 17: Classification of ASPIRE Tooling Specifications and Associated Use Cases

The above table does not include or refer to the ASPIRE privacy tool, which is described later in the document.

8.4 Privacy Framework and Tool (in accordance to Deliverable D2.5)

The ASPIRE IDE should be implemented using privacy friendly algorithms and techniques in their design. Thus, allowing for the principal of ePrivacy and other Data Protection Directives to be upheld when used to construct RFID solutions by RFID consultants. Hence, ASPIRE would be able to deliver a middleware that is privacy friendly and protects personal and sensitive data.

Management tools implemented within ASPIRE IDE would allow third parties to easily build and deploy RFID solutions with minimal programming and configuration efforts required. This would be done through simply specifying company data, processes, transactions, products, etc. In addition, the Integrated Development Environment would encompass editing tools that allow with specification and configuration of the middleware functionalities. Since the IDE manages and edits all components it is beneficial that the IDE is programmed to be capable of handling most of the privacy concerns at the software level itself.

8.4.1 Compliance with data quality principal (limiting collection of personal data)

ASPIRE aims to create a middleware that would allow building of privacy friendly RFID solutions by collecting minimalist data of sensitive or personal nature. Therefore first it is important for the system to identify what type of data is personal or sensitive and avoid collection of such data. This is done so by utilizing the master data specified by the consultants building an RFID solution, to analyse the context and ensuring privacy concerns are met by not collecting personal data or through restricting linkage between subject data and object data. For example, in the context of a pharmaceutical, products would be categorized as personal and/or sensitive and would not require after sales services (i.e. warrantee). Therefore the middleware should adopt Anonymity techniques specified in D2.5 to restrict the collection of the subjects' data to ensure the privacy of the consumers are met. On the other hand, if an application constructed is for the use of an optical store whereby recording of the consumers' data along with its transaction is necessary, the middleware should allow doing so. At the same time the middleware would trigger a privacy prompt message and log the activity through the use of flags. This would have a twofold benefit; allowing smooth and accurate auditing of the application during the

certification process, and permitting examination and correction of the process if possible by administration and/or auditors.

Upon analyzing the context, the case requires products or services entail the identification of the customer, the ASPIRE IDE should propose techniques such as allocating usernames or random ids for customers. These usernames or Ids should not be linked with any of the personal details of the consumer that could lead the identification of the person or their personal details with the knowledge of the username or id. For example, stores that would like to introduce loyalty cards to provide its customers with offers that could benefit them would be able to do so by not gathering any information of the customer and providing them with services purely based on the records of its consumptions.

8.4.2 Compliance with Data Limitation and Conservation principals

In cases where personal and/or sensitive data is stored in repositories, the ASPIRE IDE should be implemented in a way whereby it does not allow processing of the collected information for unintended purposes. This is possible by enabling ASPIRE IDE to control the access of fixed and programmable logic to these database structures. Therefore the IDE should; first control how Business Event Generator communicates to and from the Repository, second how the repository communicates and sends data to ASPIRE connector applications and thereafter. Algorithms and techniques such as encryption and vigilance of personal data as specified in Deliverable 2.5 of Privacy Specifications can further help limit the processing of data towards unintended purposes.

Furthermore, the ASPIRE IDE could also be programmed to produce cumulative statistics whereby the program would calculate statistics without registering data about individual transactions. This could further help companies perform certain market or statistical analysis using its records without endangering the privacy of its consumers. The ASPIRE IDE will also implement privacy alerts that would be triggered/logged if any programmable logic is changed to access personal data for unauthorised transactions and the ASPIRE administrator and external auditor will be immediately notified.

ASPIRE IDE would also incorporate algorithms that that would not retain and/or process personal data longer than necessary. Techniques such as include automatic cleaning mechanisms which automatically delete any personal data that is not required any more. For example once the warranty of the product is over, company does not require to hold personal details of the customer. Therefore algorithms should be defined to periodically scan data deleting personal data no longer necessary. Furthermore, ASPIRE IDE should incorporate various other algorithms specified in Deliverable 2.5 such as ‘In memory’ processing, Copy + Destroy, and Volatile Encryption to enforce conservation of data.

The privacy framework should also incorporate techniques that filter out non related data. For example solutions would not allow companies to record data that are generated through tags unrelated to the organisation. This should be implemented through appropriate configuration of the BEG. Since the BEG

applies business logic and interprets lower level events to business events from data reports generated by the Filtering and Collection layer. Algorithms could be implemented which would gather and delete data that the BEG could not reconcile with the master data provided by the companies.

8.5 ASPIRE IDE Specifications Overview

8.5.1 Management Console Specifications

The following table summarizes the ASPIRE Management Console Specifications. Implementation priority values from 1 to 5 with 5 being of most importance.

C/N	Fields to be Implemented	Priority
7.1	**Management console**	3
7.1.1	RFID readers	3
7.1.2	Reader Core proxy	3
7.1.3	F&C Server	3
7.1.4	BEG engine	3
7.1.5	EPCIS repository	3
7.1.6	Connector application	3

Table 18: Management Console Specifications requirement Overview

8.5.2 Tooling Specifications Requirement

The following table summarizes the ASPIRE Tooling Specifications. Implementation priority values from 1 to 5 with 5 being of most importance.

C/N	Fields to be Implemented	Priority
7.2.1	**ASPIRE IDE**	4
7.2.2	**Physical Reader Configuration**	3
7.2.3	**Logical Reader Configuration**	4
7.2.3.1	LLRP readers	4
7.2.3.2	RP readers	4
7.2.3.3	HAL readers and	4
7.2.3.4	Simulator readers	4
7.2.4	**Filtering Specifications Editor**	5
7.2.5	**F&C Commands Execution**	4
7.2.6	**Master Data Editor**	5
7.2.6.1	Business dispositions	5
7.2.6.2	Business steps	5
7.2.6.3	Business transactions	5
7.2.6.4	Transactions type	5
7.2.6.5	Business locations	5
7.2.6.6	Read Points	5
7.2.7	**Connector Operations**	3
7.2.8	**Workflow Management Editor**	4

Table 19: Tooling Specifications Overview

9 Conclusions

This deliverable has elaborated on the main specifications of the main functionalities of the ASPIRE middleware. In particular, it has provided specifications for a wide range of middleware modules spanning the areas of reader access, filtering and collection, automated business events generation, information sharing, as well as interfacing with legacy enterprise applications. Several of these specifications have been directly derived from EPC standards with a view to capitalizing on EPC's specification work. However, we have also outlined and specified additional middleware functionalities that extend significantly EPC's work in the area of RFID middleware. Prominent examples of such middleware functionalities lie in the area of business event generations and connectors for interfacing to legacy enterprise applications. In conjunction with Deliverables D2.5 (dealing with privacy specifications), Deliverable D3.2 (dealing also with TDT (tag data translation)) and Deliverable 3.1 (establishing the project's licensing schemes), the present deliverables establishes the main specifications for a lightweight, privacy-friendly, open-source, integrated middleware for RFID solutions.

Apart from middleware specifications, the present deliverable has elaborated on the functionality of a number of tools for facilitating integrated development, deployment and configuration of RFID solutions. These tools account for a wide range of unique features in RFID tooling, with prominent examples in the areas of business process management and integrated development. Furthermore, we have outlined the need for end-to-end infrastructure management and have specified associated (JMX based) solutions.

The specifications provided in this document manifest the both the complexity and versatility of the ASPIRE middleware platform and tools. Further to actively contributing and boosting their implementation, the ASPIRE consortium aims at involving skilful community contributors, which could engage in implementing the specifications contained in this deliverable. In this sense, this deliverable could serve as a valuable guide not only to ASPIRE developers, but also to potential contributors of the AspireRfid project. While we cannot rule out the implementation of features that are not part of this document, we think that the introduced specifications establish a sound basis for a novel and versatile middleware platform.

A next step in the evolution of the ASPIRE project, is the specification of a complete solution language that will could capture (in a declarative fashion) all the functionalities supported by the ASPIRE development and management tools. The specification of such a language, along with an associated run-time middleware engine for decoding and executing solutions written in this language, will formalise the programmability of the project. Furthermore, it will contribute to the openness of the project, through allowing third-parties to create open-source tools that can manipulate the ASPIRE solutions specifications. This is because the ASPIRE solutions language will be amenable by tools. Hence, this

deliverable paves the ground for the evolution of the ASPIRE programmability tasks, as part of WP4 of the ASPIRE project.

10 - Acronyms

ASPIRE	Advanced Sensors and lightweight Programmable middleware for Innovative Rfid Enterprise applications
BEG	Business Event Generation
BPM	Business Process Management
DCI	Discovery Configuration and Initialization
ebXML	Electronic Business using eXtensible Markup Language
EDI	Electronic Data Interchange
EPC	Electronic Product Code
EPCIS	EPC Information Services
ERP	Enterprise Resource Planning
F&C	Filtering and Collection
HAL	Hardware Abstraction Layer
HIPAA	Health Insurance Portability and Accountability Act
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
IS	Information Systems
ISO	International Standards Organization
IT	Information Technology
J2EE	Java 2 Platform Enterprise Edition
JCA	Java EE Connector Architecture
JCP	Java Community Process
J2ME	Java 2 Platform Micro Edition
JDBC	Java Database Connectivity
JMX	Java Management Extensions
JSR	Java Specification Request
JVM	Java Virtual Machine
LGPL	Lesser General Public License
LLRP	Low Level Reader Protocol
LR	Logical Reader
OSS	Open Source Software
OW2	*ObjectWeb Consortium* and *Orientware*
RFID	Radio Frequency Identification
ROI	Return Of Investment
RP	Reader Protocol
SCM	Supply Chain Management
SME	Small and Medium Enterprises
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
TCP	Transfer Control Protocol
UHF	Ultra High Frequency
UML	Unified Modeling Language
URI	Uniform Resource Identificator
WSDL	Web Service Definition Language
WMS	Warehouse Management System
XML	Extensible Markup Language

11 List of Figures

12 List of Tables

13 - 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, "Reader Management 1.0.1, May 31", 2007 available at: http://www.epcglobalinc.org/standards/rm
[6] EPCglobal, "EPCglobal Tag Data Standards, Version 1.4", June 11, 2008, available at: http://www.epcglobalinc.org/standards/tds/
[7] EPCglobal, "EPCglobal Tag Data Translation (TDT) 1.0", January 21, 2006 available at: http://www.epcglobalinc.org/standards/tdt/
[8] EPC Information Services (EPCIS) Specification, Version 1.0.1, September 21, 2007 available at: http://www.epcglobalinc.org/standards/epcis/
[9] LLRP Toolkit, http://www.llrp.org/
[10] 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.
[11] C.Floerkemeier, C. Roduner, and M. Lampe, RFID Application Development With the Accada Middleware Platform, IEEE Systems Journal, Vol. 1, No. 2, December 2007.
[12] 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.
[13] Russell Scherwin and Jake Freivald, Reusable Adapters: The Foundation of Service-Oriented Architecture, 2005.
[14] The XMOJO Project Product Documentation, available at: http://www.jmxguru.com/products/xmojo/docs/index.html
[15] Java Management Extensions (JMX) Technology Overview, available at: http://java.sun.com/j2se/1.5.0/docs/guide/jmx/overview/architecture.html
[16] Panos Dimitropoulos and John Soldatos, 'RFID-enabled Fully Automated Warehouse Management: Adding the Business Context', submitted to the International Journal of Manufacturing Technology and Management (IJMTM), Special Issue on: "AIT-driven Manufacturing and Management".
[17] Architecture Review Committee, "The EPCglobal Architecture Framework," EPCglobal, July 2005, available at: http://www.epcglobalinc.org.
[18] Achilleas Anagnostopoulos, John Soldatos and Sotiris G. Michalakos, 'REFiLL: A Lightweight Programmable Middleware Platform for Cost Effective RFID Application Development', accepted for publication to the Journal of Pervasive and Mobile Computing (Elsevier).
[19] Application Level Events 1.1(ALE 1.1) Overview, Filtering & Collection WG, EPCglobal, March 5, 2008 , available at: http://www.epcglobalinc.org/standards/ale
[20] WS-I, Basic Profile v1.0, available at: http://www.ws-i.org/Profiles/BasicProfile-1.0-2004-04-16.html.

[21] Benita M. Beamon, “Supply chain design and analysis: Models and methods”, International Journal of Production Economics, Vol. 55 pp. 281-294, 1998

[22] Zhekun Li, Rajit Gadh, and B. S. Prabhu, "Applications of RFID Technology and Smart Parts in Manufacturing", Proceedings of DETC04: ASME 2004 Design Engineering Technical Conferences and Computers and Information in Engineering Conference September 28-October 2, 2004, Salt Lake City, Utah USA.

[23] JSR 256, “Mobile Sensor API”, available at: http://jcp.org/en/jsr/detail?id=256

[24] JSR 275, “Units Specification”, available at: http://jcp.org/en/jsr/detail?id=275

[25] JSR 179, “Location API for J2ME”, available at: http://jcp.org/en/jsr/detail?id=179

[26] JSR 257, “Contactless Communication API”, available at: http://jcp.org/en/jsr/detail?id=257

Appendix A - Example of Business Event and Processes in Warehouse Management)

In this section we illustrate the concept of business events for key warehouse management processes, based on the EPCIS framework. We focus on common processes such as receiving, shipping, moving, pick & pack, as well as inventory and envisage that the provided descriptions could be reusable across multiple enterprises. Note that we currently emphasize on these processes in the scope of closed loop enterprise systems concerning a single enterprise and intra-enterprise transactions, rather than open loop systems spanning multiple business partners and cross enterprise transactions. Our starting point is to formulate the company's structure and warehouses, which is a key prerequisite to populating user-created identifiers (e.g., read points and business locations) of the EPC-IS framework.

A.1 Taxonomy of Warehouses and Containers

Consider a typical enterprise possessing a number of logical spaces identified as Warehouses (Wn (n = 0, 1, 2,…)). We also assume that these warehouses are organized in a hierarchical manner in a way that each warehouse is contained within another warehouse or equivalently each warehouse has one parent warehouse. We define as W0 the (logical) central warehouse of the company, which has no parent warehouse. Therefore, all warehouses can be collectively aggregated under W0, which can be considered as a physical central warehouse or the company itself. Moreover, child logical warehouses may correspond to physical warehouses or other units of storing capacity down the hierarchy (e.g., selves that are contained within a physical warehouse space).

Warehouse management processes associated with the company's products are carried out based on appropriately tagged containers (Cn (n = 0, 1, 2,…,)). Different types of containers are typically used e.g., pallets, carton boxes, carts, containers. Similarly to logical warehouses containers are organized in an hierarchical fashion, which allows containers (e.g., pallets) to contain other containers (e.g., carton boxes). Furthermore, a container is situated to a parent logical warehouse. A container (Cn) is contained in a warehouse, as soon as this warehouse contains a parent container of (Cn), which allows us to infer locations for child containers.

Since both containers and logical warehouses can contain other containers and/or items, it makes sense to distinguish between container and logical warehouses. The key difference is that when items within a container move, the container moves as well, whereas when items within a logical warehouse move, the logical warehouse does not move. Likewise, one logical warehouse has typically one parent object (i.e. the parent warehouse), while a container has typically two parent objects (i.e. a parent warehouse and a parent container).

A.2 Examples of Elementary Business Processes

A2.1 Receiving

An indicative RFID-enabled reception of goods process within a warehouse W_R is described through the EPCIS events listed in Table 20, Table 21, Table 22 and Table 23. It is assumed that W_R is equipped with (one or more) RFID dock door portals. Items and containers pass through these portals in the scope of the "receiving" process. The process starts with a transaction event, which signifies the commencement of the receiving process and assigns an identifier to the transaction instance (BT_n) (Table 20). Note that this identifier enables the connection of the transaction with expected goods (listed within the company's WMS) for this particular receiving process. The transaction event will be inserted into the EPCIS repository, prior to the appearance of goods. The transaction start event will be typically associated with a dispatch (consignment) note (or delivery note) regarding the expected items.

Subsequent events denote that objects are received within the warehouse. Objects correspond to items, as well as containers. An object event is issued to identify the received objects, while a transaction event is also created to denote that the "*receiving*" transaction is in progress (Table 21). The event includes also information about the Business Location i.e. the warehouse (e.g., $W_{R)}$), where the reception of goods takes place. Furthermore, the transaction event denotes the status of the received goods, as well as the related business step where these good were observed during receiving.

Event Description			
EventType	**Time**	**bizStepID**	**dispositionID**
TransactionEvent	Time	Null	D_n
BizLocationID	**readPointID**	**EPC**	**parentEPC**
Null	Null	<EPC List>	null
Action	**bizTransactionTypeID**	**bizTransactionID**	
Add	Null	BT_n	

Table 20: Creation of "Receiving" Transaction (TS Event)

Event Description			
eventType	**Time**	**bizStepID**	**dispositionID**
ObjectEvent	Time	Null	null
bizLocationID	**readPointID**	**EPC**	**parentEPC**
W_R	Null	<EPC List>	null
Action	**bizTransactionTypeID**	**bizTransactionID**	
add	Null	Null	
Event Description			
eventType	**Time**	**bizStepID**	**dispositionID**

TransactionEvent	Time	D_n	D_m
bizLocationID	**readPointID**	**EPC**	**ParentEPC**
null	Null	<EPC List>	null
Action	**bizTransactionTypeID**	**bizTransactionID**	
observed	Null	BT_n	

Table 21: Creation of Objects in the Warehouse during "Receiving" and association with the running transaction (i.e. "observing the transaction)

Table 22 describes the aggregation event which is accordingly issued in order to associate items with containers during the "receiving" process. The aggregation event identifies also the business location and the read-point, where the aggregation of objects into containers tools place. Note that additional aggregations can occur (e.g., aggregation of carton boxes to pallets). To capture these aggregations based on an RFID-enabled system, the pallets need to pass from an additional RFID dock-door portal, which can enable the issuance of additional aggregation events. This two level physical process (i.e. aggregation into boxes and subsequent aggregation into pallets) is therefore totally reflected in the issued aggregation events.

Event Description			
eventType	**Time**	**bizStepID**	**dispositionID**
AggregationEvent	Time	Null	Null
bizLocationID	**readPointID**	**EPC**	**parentEPC**
W_R	W_R	<EPC List>	EPC
Action	**bizTransactionTypeID**	**bizTransactionID**	
Add	Null	Null	

Table 22: Objects Aggregation in appropriate containers within the Warehouse during "Receiving"

The "receiving" process is concluded when all items (assigned to this particular transaction) have entered W_R via one of the available RFID dock door portals. At this point the RFID system will issue a transaction finish event, which will contain the full list of received items. Note that this may or may not be the same list specified within the transaction start events. Items expected but not received will not be reported. The transaction finish event will be accompanied by the issuance of a delivery receipt from the WMS system, based on the information contained in the EPCIS repository.

Event Description			
eventType	**Time**	**bizStepID**	**dispositionID**
TransactionEvent	Time	Null	null
bizLocationID	**readPointID**	**EPC**	**parentEPC**
Null	Null	<EPC List>	null
Action	**bizTransactionTypeID**	**bizTransactionID**	

Delete	Null	BT_n	

Table 23: Closing the "Receiving" Transaction

A2.2 Moving within Logical Warehouses

The process of tracking items and correctly recording their state as they move within the company's warehouse relies on the logical partitioning of the company into multiple warehouses. Typically, goods that are moved between logical warehouses of the company have been gracefully received previously and are considered part of W_0. During the moving procedure it is important to understand whether an object enters or leaves a warehouse through an RFID dock-door portal. To this end, the lower layers of the RFID system (e.g., the physical readers) must also report the directionality of the "moving" process for a particular item (i.e. "in" or "out"). An object identified to move out of a logical warehouse during the "move", may either be detected to enter another logical warehouse, or may just remain associated with the parent logical warehouse W_0. Note also that moving goods between selves or carts is possible. Such movement is likely to be detected by mobile readers rather than dock-door portals.

Events that can be used to add business context to the "moving" process are depicted in Table 24, Table 25, Table 26, Table 27, Table 28, Table 29, Table 30, which add the business context of the key "moving" processes. In particular, Table 24 described the object event that is issued when an item is moved out of a logical warehouse (e.g., for packaging). It covers also the case where an item is removed from a self. Similarly, Table 25 conveys the dual process, i.e. moving an item towards a logical warehouse from another logical warehouse, which is higher in the hierarchy.

Event Description			
eventType	**Time**	**bizStepID**	**dispositionID**
ObjectEvent	Time	Null	Null
bizLocationID	**readPointID**	**EPC**	**parentEPC**
W_0, W_m	W_n	<EPC List>	Null
Action	**bizTransactionTypeID**	**bizTransactionID**	
Observed	Null	Null	

Table 24: Event for Objects leaving W_n and entering W_0 or its parent W_m (e.g., when W_n is a self)

Event Description			
eventType	**Time**	**bizStepID**	**dispositionID**
ObjectEvent	Time	Null	Null
bizLocationID	**readPointID**	**EPC**	**ParentEPC**
W_n	W_m, W_0	<EPC List>	Null
Action	**bizTransactionTypeID**	**bizTransactionID**	
Observed	Null	Null	

Table 25: Event for Objects entering W_n from W_0 or its parent W_m (e.g., assuming that W_n is a self)

Table 26 and Table 27 illustrate aggregation events that have to be issued in case when an item is moved within a container. Specifically, Table 26 refers to the case when a container leaves a logical warehouse for another warehouse, whereas Table 27 describes the event that has to be generated by the RFID System in cases when a container is inserted into the target logical warehouse. In both cases aggregation observed events are generated.

Event Description			
EventType	**Time**	**bizStepID**	**dispositionID**
AggregationEvent	Time	Null	Null
bizLocationID	**readPointID**	**EPC**	**parentEPC**
W_0, W_m	W_n	<EPC List>, null	null, EPC
Action	**bizTransactionTypeID**	**bizTransactionID**	
Observed	null	Null	

Table 26: Event for Aggregated Objects leaving W_n and entering W_0 or its parent W_m (e.g., when W_n is a self)

Event Description			
eventType	**Time**	**bizStepID**	**dispositionID**
AggregationEvent	Time	null	Null
bizLocationID	**readPointID**	**EPC**	**parentEPC**
W_n	W_0, W_m	<EPC List>, null	null, EPC
Action	**bizTransactionTypeID**	**bizTransactionID**	
Observed	Null	Null	

Table 27: Event for Aggregated Objects entering W_n from W_0 or its parent W_m (e.g., assuming that W_n is self)

The "moving" process requires issuance of events that assign business context in cases where items are taken out of a container (Table 28), as well as in cases where objects are packaged in the container (Table 29). In these cases aggregations have to be deleted and created respectively. Finally, Table 30 illustrates the event where a container C_n is destroyed in which case another aggregation event has to be issued.

Event Description			
eventType	**Time**	**bizStepID**	**dispositionID**
AggregationEvent	Time	Null	null
bizLocationID	**readPointID**	**EPC**	**parentEPC**
Null	null	<EPC List>	C_n
Action	**bizTransactionTypeID**	**bizTransactionID**	
Delete	Null	Null	

Table 28: Event for taking Objects out of a Container

Event Description			
eventType	**Time**	**bizStepID**	**dispositionID**
AggregationEvent	Time	Null	null
bizLocationID	**readPointID**	**EPC**	**parentEPC**
Null	null	<EPC List>	C_n
Action	**bizTransactionTypeID**	**bizTransactionID**	
Add	null	Null	

Table 29: Event for packing Objects within a Container

Event Description			
eventType	**Time**	**bizStepID**	**dispositionID**
AggregationEvent	Time	Null	null
bizLocationID	**readPointID**	**EPC**	**parentEPC**
Null	Null	Null	C_n
action	**bizTransactionTypeID**	**bizTransactionID**	
delete	Null	Null	

Table 30: Event for Destroying Container C_n

A2.3 Order Collection - Pick & Pack

Another common warehouse management process concerns the order collection (pick & pack). We consider a typical pick & pack process that hinges on the existence of an order note. Based on the order note an order collection note comprising the ordered item is also issued. The order collection note is associated with a transaction start event (**Table 31**). The order collection note list the individual items contained in the order. Moreover, it can also contain container codes, in the case where the items in the order are entirely contained within a container.

Event Description			
eventType	**Time**	**bizStepID**	**dispositionID**
TransactionEvent	Time	Null	D_n
bizLocationID	**readPointID**	**EPC**	**parentEPC**
Null	Null	<EPC List>	null
Action	**bizTransactionTypeID**	**BizTransactionID**	
Add	Null	BT_n	

Table 31: Transaction Event signifying the commencement of the pick & pack process for an order

The order collection occurs within portable containers (e.g., carts). Objects are removed from the selves (**Table 32**) and aggregated to the carts (**Table 33**). Note

that during this collection processes it is possible to automatically check for errors e.g., in cases where an object that is not listed in the order note is removed from the self. As already outlined except for individual items, the order collection process may pick whole containers (e.g., boxes), as soon as these are part of the order collection process. This is denoted by an aggregation observed event (Table 34). Furthermore, Table 35 depicts a new aggregation event denoting that a whole package is put within the portable container during the order collection process.

Event Description			
eventType	**Time**	**bizStepID**	**dispositionID**
ObjectEvent	Time	Null	Null
bizLocationID	**readPointID**	**EPC**	**parentEPC**
W_m	W_n	<EPC List>	Null
Action	**bizTransactionTypeID**	**BizTransactionID**	
Observed	Null	Null	

Table 32: Object Event denoting that an object is moved from a shelf

Event Description			
eventType	**Time**	**bizStepID**	**dispositionID**
AggregationEvent	Time	Null	Null
bizLocationID	**readPointID**	**EPC**	**parentEPC**
Null	Null	<EPC List>	C_n
Action	**bizTransactionTypeID**	**BizTransactionID**	
Add	Null	Null	

Table 33: Aggregation Event denoting that an object is moved to a portable container (e.g., cart)

Table 36 and Table 37 describe transaction events that are issued to declare the progress of the collection process. These denote the items (Table 36), as well as the whole containers (Table 37) that are already collected and ready to be shipped in the scope of the pick & pack process. Finally, Table 38 and Table 39 describe aggregation events issued as individual items or whole containers are moved out of the originating warehouse (to W_0) and from W_0 to the shipping warehouse (W_s) respectively.

Event Description			
EventType	**Time**	**bizStepID**	**dispositionID**
AggregationEvent	Time	Null	Null
bizLocationID	**readPointID**	**EPC**	**parentEPC**
W_m	W_n	null	C_m
Action	**bizTransactionTypeID**	**bizTransactionID**	
Observed	Null	Null	

Table 34: Aggregation Event for objects packed in (whole containers)

Event Description			
eventType	**Time**	**bizStepID**	**DispositionID**
AggregationEvent	Time	Null	Null
bizLocationID	**readPointID**	**EPC**	**ParentEPC**
Null	Null	C_m	C_n
Action	**bizTransactionTypeID**	**bizTransactionID**	
Add	Null	Null	

Table 35: Aggregation Event for placing objects in packed container C_m to a portable container C_n (e.g., cart)

Event Description			
eventType	**Time**	**bizStepID**	**dispositionID**
TransactionEvent	Time	D_m	D_n
bizLocationID	**readPointID**	**EPC**	**parentEPC**
null	Null	<EPC List>	Null
Action	**bizTransactionTypeID**	**bizTransactionID**	
observed	Null	BT_n	

Table 36: Transaction Event denoting that an object of the order is ready to be dispatched

Event Description			
eventType	**Time**	**bizStepID**	**dispositionID**
TransactionEvent	Time	D_m	D_n
bizLocationID	**readPointID**	**EPC**	**parentEPC**
Null	Null	null	C_m
Action	**bizTransactionTypeID**	**bizTransactionID**	
Observed	Null	BT_n	

Table 37: Transaction Event denoting that a group of objects (C_m) of the order is ready to be dispatched

Event Description			
eventType	**Time**	**bizStepID**	**dispositionID**
AggregationEvent	Time	Null	Null
bizLocationID	**readPointID**	**EPC**	**parentEPC**
W_0	Null	<EPC List>	Null
Action	**bizTransactionTypeID**	**bizTransactionID**	
Observed	Null	null	

Table 38: Aggregation Event denoting that a group of items are moved to W_0

Event Description			
EventType	**Time**	**bizStepID**	**dispositionID**
AggregationEvent	Time	Null	null
BizLocationID	**readPointID**	**EPC**	**parentEPC**
W_S	W_0	<EPC List>	null
Action	**bizTransactionTypeID**	**bizTransactionID**	
Observed	Null	Null	

Table 39: Aggregation Event denoting that a group of items are moved from W_0 to warehouse W_s

A2.4 Order Shipment

The events that provide the business context for the order shipment process are depicted in Table 40, Table 41, Table 42, Table 43, Table 44, and Table 45. This business process takes place in the scope of the "shipping" warehouse (W_S), where products that have to be shipped are assembled. Shipping hinges on logistically moving items and containers (with items) out of warehouse W_S. Based on the order collection process items and containers have been put within carts. In the scope of the order shipment process these aggregations are deleted (Table 40, Table 41) i.e. items and containers are moved out of the pick & pack carts. Accordingly, new aggregation events (Table 42) signifying the creation of packing lists for the shipment process. Once packing lists are complete, the objects are moved out of the warehouse (W_S), which is signified through object delete events (i.e. the objects are no longer in the warehouse) (Table 43). Also transaction events are issued to convey and control the status of the process. In particular, transaction observed events (Table 44) provide insight on the objects that have been shipped, whereas a transaction finish event (Table 45). During the issuance of the transaction finish event, the system can automatically check whether the packing list coincides with the shipment list, which signifies the graceful completion of the order shipment process.

Event Description			
EventType	**Time**	**bizStepID**	**dispositionID**
AggregationEvent	Time	Null	Null
bizLocationID	**readPointID**	**EPC**	**parentEPC**
Null	Null	<EPC List>	C_n
Action	**bizTransactionTypeID**	**bizTransactionID**	
Delete	Null	null	

Table 40: Aggregation Event denoting that objects are moved out of the cart (C_n) (i.e. aggregation deleted)

Event Description

EventType	**Time**	**bizStepID**	**dispositionID**
AggregationEvent	Time	null	null
bizLocationID	**readPointID**	**EPC**	**parentEPC**
Null	Null	C_m	C_n
Action	**bizTransactionTypeID**	**bizTransactionID**	
Delete	Null	Null	

Table 41: Aggregation Event denoting that a whole group of objects (C_m) (e.g., package) are moved out of the cart (C_n) (i.e. aggregation deleted)

Event Description			
EventType	**Time**	**bizStepID**	**dispositionID**
AggregationEvent	Time	null	null
bizLocationID	**readPointID**	**EPC**	**parentEPC**
W_S	W_S	<EPC List>	C_n
Action	**bizTransactionTypeID**	**bizTransactionID**	
Add	Null	null	

Table 42: Packaging of objects within a container (C_n)

Event Description			
EventType	**Time**	**BizStepID**	**dispositionID**
ObjectEvent	Time	Null	null
bizLocationID	**readPointID**	**EPC**	**ParentEPC**
Null	W_S	<EPC List>	null
Action	**bizTransactionTypeID**	**bizTransactionID**	
Delete	Null	Null	

Table 43: Objects leaving the Warehouse where W_S where the shipment is conducted (i.e. Object Event Delete)

Event Description			
EventType	**Time**	**bizStepID**	**dispositionID**
TransactionEvent	Time	D_m	D_n
bizLocationID	**readPointID**	**EPC**	**parentEPC**
Null	Null	<EPC List>	Null
Action	**bizTransactionTypeID**	**bizTransactionID**	
Observed	Null	BT_n	

Table 44: Transaction Event for Objects that have been shipped

Event Description			
eventType	**Time**	**bizStepID**	**dispositionID**
TransactionEvent	Time	D_n	Null
bizLocationID	**readPointID**	**EPC**	**parentEPC**
Null	null	<EPC List>	Null
Action	**bizTransactionTypeID**	**bizTransactionID**	
Delete	Null	BT_n	

Table 45: Transaction event for concluding the order shipment process

A2.5 Inventory

Inventory is a very common and important process for warehouse management. RFID enabled automated inventory is (at the EPCIS level) carried out based on an object event listing all the objects that are scanned and/or detected in the warehouse selves (e.g., via mobile readers) (Table 46). Note that object observe events having a bizLocationID identical to the readPointD signify that objects are simply observed and not moved in the same warehouse. Note also that a transaction start (Table 47), as well as a transaction finish (Table 48) events provide identity to the inventory process and specify its boundaries. The transaction start event includes also the list (<EPC List>) of expected items, whereas the transaction finish event includes the list (<EPC List>) of actually observed items.

Event Description			
EventType	**Time**	**bizStepID**	**dispositionID**
ObjectEvent	Time	null	Null
bizLocationID	**readPointID**	**EPC**	**parentEPC**
W_n	W_n	<EPC List>	Null
Action	**bizTransactionTypeID**	**bizTransactionID**	
Observe	Null	Null	

Table 46: Automated Inventory of all objects within a warehouse W_n

Event Description			
eventType	**Time**	**bizStepID**	**dispositionID**
TransactionEvent	Time	Null	D_n
bizLocationID	**readPointID**	**EPC**	**parentEPC**
null	Null	<EPC List>	null
action	**bizTransactionTypeID**	**bizTransactionID**	
add	Null	BT_n	

Table 47: Transaction event denoting the commencement of the inventory (and the expected situation)

Event Description			
eventType	**Time**	**bizStepID**	**dispositionID**
TransactionEvent	Time	D_n	Null
bizLocationID	**readPointID**	**EPC**	**parentEPC**
null	Null	<EPC List>	Null
action	**bizTransactionTypeID**	**bizTransactionID**	
delete	Null	BT_n	

Table 48: Transaction event denoting the completion of the inventory (and the reported situation)

A.3 Complex Business Process

In order to provide a more general framework for handling RFID enabled logistics possesses this deliverable suggests encoding the above events within a processes description that could be amenable by graphical tools. To this end, it is possibly to use the master data concept in order to encode business locations, read points, logical warehouses, containers, disposition states, as well as event and business steps sequences. Having a business processes encoded as master data, it could be possible to offer to business users and RFID consultants a graphical user interface for manipulating the definition of the various business transactions and their associated business steps.

The logistics processes outlined above can operate in a totally independent fashion, which provides opportunities for incremental deployment and smooth transition from legacy manual non-RFID processes. Incremental deployment can lower a company's entry costs, while boosting its understanding and experience with respect to RFID technology. Note however that the implementation of all the processes, along with their integration (e.g., connecting the pick & pack with the shipment process) can enable higher degrees of automation and efficiency, overall yielding a multiplicative benefit. Hence, moving at a higher business processes management level, it is possible to define how individual business transactions and processes are connected and/or integrated in the scope of wider process. Such higher level Business Processes Management (BPM) can be supported by conventional BPM tools, as described in the ASPIRE BPM framework that has been illustrated in this deliverable.