

Collaborative Project

ASPIRE

Advanced Sensors and lightweight Programmable
middleware for Innovative Rfid Enterprise applications

FP7 Contract: ICT-215417-CP**WP3 – RFID Middleware Infrastructure****Public report - Deliverable****Readers and Tags Virtualization**

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

Deliverable ID:	WP3/D3.2
Deliverable Title:	Readers and Tags Virtualization
Responsible partner:	Athens Information Technology (AIT)
Main Contributors:	John Soldatos (AIT) Nikos Kefalakis (AIT) Nektarios Leontiadis (AIT) Dimitris Metafas (AIT) Denis Ruffieux (MELEXIS) Nathalie Mitton (INRIA) Loïc Schmidt (INRIA) Didier Donsez (UJF)
Estimated Indicative Person Months:	10

Start Date of the Project: 1 January 2008

Duration: 36 Months

Revision: 1.0

Dissemination Level: PU

PROPRIETARY RIGHTS STATEMENT

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

Document Information

Document Name: Readers and Tags Virtualization
Document ID: WP3/D3.2
Revision: 1.0
Revision Date: 30 September 2008
Author: Athens Information Technology (AIT)
Security: PU

Approvals

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

Reviewers

Name	Organization	Date	Comments	Visa
Patrick Albert	MELEXIS	23-09-08		
Mathieu DAVID	AAU	22-09-08		
Julien VINAY	PV	22-09-08		

Document history

Revision	Date	Modification	Authors
0.1	11 July 08	First draft	John Soldatos, Nikos Kefalakis, Nektarios Leontiadis
0.2	30 July 08	Added ASPIRE Reader Interfaces	Nikos Kefalakis
0.3	01 Sep 08	Introduction, Readers & Tags Virtualization In ASPIRE Architecture, Added ASPIRE Low-Cost reader	John Soldatos, Nikos Kefalakis, Didier Donsez
0.4	12 Sep 08	Tag and bar codes Data Standards, tag data translation in ASPIRE	Nathalie Mitton, Loïc Schmidt
0.5	18 Sep 08	Revisions; Added Conclusions and Executive Summary; Document Sent to (Internal) Reviewers	John Soldatos, Dimitris Metafas, Denis Ruffieux
0.6	26 Sep 08	Added/Accepted reviews and comments, Minor Corrections/Additions	Nikos Kefalakis

0.7	26 Sep 08	Added planned tag translations	Nikos Kefalakis, Nathalie Mitton, Loïc Schmidt
1.0	30 Sep 08	Final Version, Fine Tuning	Nikos Kefalakis, John Soldatos

Table of Contents

Executive Summary 5

Section 1 - Introduction..... 7

Section 2 - Readers & Tags Virtualization Overview..... 8

2.1 Reader Virtualization 8

2.2 Tag Virtualization..... 8

Section 3 - ASPIRE Reader Interfaces 9

3.1 Low Level Reader Protocol Interface (LLRP)..... 9

3.2 Reader Protocol Interface (RP)..... 10

3.3 Hardware Abstraction Layer Interface (HAL)..... 11

 3.3.1 Rationale behind using HAL 12

 3.3.2 HAL Architecture..... 12

 3.3.3 Reader Core Proxy 13

 3.3.3.1 Architectural Overview 13

 3.3.3.1.1 Configuring the Reader..... 14

 3.3.4 Various Vendors Reader Protocols 15

3.4 ASPIRE Classical & Low-Cost reader..... 15

Section 4 - Tag Data Standards 17

4.1 EPC global Tag Data Standards 17

 4.1.1 GID 17

 4.1.2 SGTIN 17

 4.1.3 SSCC 18

 4.1.4 SGLN 18

 4.1.5 GRAI 19

 4.1.6 GIAI..... 19

 4.1.7 DoD..... 19

4.2 ISO Tag Data Standards..... 19

 4.2.1 ISO 15961 19

 4.2.2 ISO 15962..... 20

 4.2.3 ISO 15963..... 20

 4.2.4 ISO 15693..... 20

 4.2.5 ISO 14443..... 20

4.3 GS1 Bar Code Data Standards 20

Section 5 – ASPIRE Tag Data Translation Implementation 22

5.1 From Barcode to EPC TDS 22

5.2 From ISO to EPC TDS..... 22

5.3 From Various EPC Formats to other EPC Formats 22

5.4 ASPIRE Tag Data Translation.....23
5.5 Tag Data Translation Roadmap.....24
Section 6 - Conclusions25
Section 7 - Acronyms26
List of Figures.....27
Section 8 - References and bibliography28

Executive Summary

ASPIRE is developing an innovative royalty free middleware platform. This middleware platform is a primary target of the open source “*AspireRfid*” project, which has been recently established in the scope of the OW2 community. The open nature of the “*AspireRfid*” project, asks for versatility in terms of the hardware that will support the RFID solutions, which will be built based on the ASPIRE middleware platform. In principle the ASPIRE middleware should be able to operate with any reader platform regardless of vendors, frequency and supported functionality. Likewise, the ASPIRE middleware should support different tag formats. This freedom of choice is perfectly in line with both the “open” nature of the middleware and the requirements of the Small Medium Enterprise (SMEs). Avoiding vendor and technology lock-in is a major requirement from the SME community with respect to RFID solutions. As a result, the ASPIRE middleware incorporates reader and tag virtualization capabilities, which does not rule out any reader or tag from being used with the ASPIRE middleware.

The present deliverable presents the reader and tag virtualization solution of the ASPIRE middleware. In the area of reader virtualization, the deliverable recommends the use of standard protocols (such as EPC-RP and EPC-LLRP) for communicating with readers in a vendor independent fashion. These protocols allow upstream middleware layers (e.g., the filtering & collection (F&C) layer specified in the ASPIRE architecture) to communicate with readers based on a set of abstract vendor independent commands. For readers that are not directly compliant to the above standard protocols, a specialized middleware layer (called Hardware Abstraction Layer (HAL) undertakes to map the abstract vendor independent commands to the low-level capabilities of the target interrogator. In this way, readers featuring different functionalities and operating at different frequencies can interface to the ASPIRE middleware, as soon as an appropriate HAL is implemented. In the scope of this deliverable we have implemented several HALs corresponding to different readers. The deliverable puts also special emphasis in illustrating how the ASPIRE middleware is interfaced to the ASPIRE reader (developed in WP5), using the mechanisms outlined above. We envisage that the list of supporting readers will continue to grow during the ASPIRE project’s lifecycle. Community developers engaging in the development and evolution of the “*AspireRfid*” project are expected to contribute new HAL implementations, which will be added on top of current implementations. In the scope of this deliverable we describe the HAL approach, along with the mechanisms that enable the interconnection of the standard abstract reader access commands to upstream middleware modules (notably the F&C layer). A proof-of-concept implementation of these mechanisms is contained in a CD accompanying this deliverable. This implementation will be incorporated in the “*AspireRfid*” codebase.

In addition to virtualized reader access, this deliverable elaborates on the ability of the ASPIRE middleware to support multiple tags. A number of tag translations are described and implemented based on appropriate connector modules residing between the reader access middleware and the RFID tag. Similar to the readers’ case, the number of supported tag formats and associated connectors will most likely grow steadily during the ASPIRE project evolution. In the scope of this evolution we have also prioritized the support of tag formats that are extensively used outside EU (a prominent example being uCode in Japan). Note that the ASPIRE middleware will also provide support for legacy bar-codes, since this is still a requirement for many automatic identification applications. Furthermore, bar code support is among the requirements that have been articulated by SMEs in the scope of the ASPIRE user requirements process (as illustrated in ASPIRE Deliverable D2.2).

The mechanisms and libraries developed in this deliverable will soon become an integral and critical part of the “*AspireRfid*” middleware. By and large, this deliverable has established

reader and tag virtualization mechanisms that guarantee extensibility and technological longevity of the ASPIRE solutions (in terms of the RFID hardware that will support them).

Section 1 - Introduction

The main goal of the ASPIRE project [16] is develop a royalty free, open source, programmable middleware platform for building RFID solutions. This platform is expected to facilitate European companies in general and SMEs in particular to develop, deploy and evolve RFID solutions. In-line with its open-source nature this platform aims at offering immense flexibility and maximum freedom to potential developers and deployers of RFID solutions. This versatility includes the freedom of choice associated with the RFID hardware (notably tags and interrogators), which will support the solution. Hence, RFID middleware is designed and implemented with a view to being totally independent from particular readers or tags. It will therefore support different reader and tag types from different vendors. The ASPIRE RFID solutions might therefore engage readers that operate at different frequencies (e.g., High Frequency, Ultra High Frequency), and feature differences in their supported functionality. This is achieved through a process, which is called “reader and tags virtualization” and is described in this deliverable.

Hence, this deliverable presents in detail the ability of the ASPIRE middleware to interface with multiple heterogeneous multi-vendor readers, as well as to support various tag types. The deliverable consists of a prototype implementation of the ASPIRE Tag Data Translation (TDT) and reader virtualization concepts, as well as an associated report documenting the prototype. The present document is the report, whereas the implementation is contained on a CD accompanying this report.

In addition to an introductory (Section 1), an overview (Section 2) and a conclusions (Section 6) section, the report is structured into two distinct parts:

- The first part (Section 3) focuses on the ASPIRE Reader virtualization and especially at the interfaces that are implemented to accomplish it. More specifically:
 - Paragraph 3.1 presents the first communication interface between ASPIRE architecture and an RFID reader the Low Level Reader Protocol Interface (LLRP)
 - Paragraph 3.2 presents the second communication interface between ASPIRE architecture and an RFID reader the Reader Protocol Interface (RP).
 - Paragraph 3.3 presents the last communication interface between ASPIRE architecture and an RFID reader which is the Hardware Abstraction Layer Interface (HAL). More specifically it presents its architecture and justifies the reason why it is needed. Moreover it presents the Reader Core Proxy architectural Overview which embeds HAL and how it is configured. Finally it presents some Reader Protocols which various RFID vendors are using that already have a HAL support. Moreover the HAL layer for the ASPIRE low cost reader is illustrated.
- The second part (comprising sections 4-5) focuses on the Tag virtualization, in particular:
 - Section 4 introduces the Tag Data Standards and focusing on the EPC global Tag Data Standards and the ISO Tag Data Standards.
 - Section 5 focuses on the implementation roadmap for the Tag Data Translation and especially from Barcode to EPC TDS, from ISO to EPC TDS and from Various EPC Formats to other EPC Formats.

Section 2 - Readers & Tags Virtualization Overview

2.1 Reader Virtualization

All RFID reader vendors equip their products with a proprietary communication interface protocol. In many cases, even readers from the same vendor are equipped with a different protocol. Thus, communicating with a reader, poses a need for becoming familiar with the reader's functionalities and specifications, while also understanding its low protocols (i.e. messages and commands). This situation becomes much more complicated, in the case of applications comprising many RFID readers from different vendors.

The role of the reader virtualization layer is to unify the way we interact with the miscellaneous hardware, by inserting a hardware abstraction layer and providing a standard instruction set to the higher layers, which require information from the hardware. This is an early identified and well understood problem in RFID systems (see for example [9], [10], [12], [13], [14]), including emerging applications (e.g., [15]). Moreover, reader virtualization is a functionality addressed by several RFID middleware implementations (see for example [11] and references therein).

Specifications that satisfy the need for a norm at the reader access level already exist. In particular, the EPCglobal Reader Protocol (RP) [4] and the EPCglobal Lower Level Reader Protocol (LLRP) [3] provide standard ways for interacting with RFID readers. These protocols define the standard bindings through which an application can send messages in a standardized format. Towards achieving reader virtualization a Hardware Abstraction Layer (HAL), ensuring a graceful mapping of the standardized messages to the low-level vendor specific reader communication primitives is specified. The methods of communication between the HAL and the hardware itself will vary, depending on the hardware vendor and it may require a serial connection, an Ethernet connection, etc. The transport protocols of communication may also vary from a raw TCP connection, to SSL and HTTP. Command and message encodings may also vary (e.g., text, XML or binary).

2.2 Tag Virtualization

Tag virtualization capitalizes on a machine-readable version of the EPC Tag Data Standards specification. This machine-readable version can be used for abstracting underlying RFID tags through bridging and mapping different representations. Hence, a tag translation module is a (standalone or embedded) middleware component that enables the interpretation of machine readable version of the tag.

The interpretation can be used in automated fashion. In addition the tag translation specification can be used for validating machine readable formats of the EPC tags. The TDT engine built in the scope of the AspireRfid project [17] supports the ISO15693, ISO14443, ISO15961, ISO15962, ISO15963, various GS1 formats (EAN/UPC, GS1 DataBar, GS1-128, ITF-14, GS1 DataMatrix, and Composite Component), as well as Bar Codes 1D and 2D: Note that barcode support is deemed particularly important given the vast number of legacy barcode applications, which need to be interoperable with emerging RFID applications.

Section 3 - ASPIRE Reader Interfaces

One important feature of the ASPIRE middleware platform is the ability to connect with multiple RFID or even Barcode readers. In Figure 1 below depicts reader interfaces that are already supported by the ASPIRE middleware. These interfaces have all been implemented in the scope of the prototype accompanying this deliverable. Note that the various readers are accessed via the EPC-RP and EPC-LLRP protocols [3-4], from upstream layers of the ASPIRE middleware. In Figure 1 it is shown how different readers connect to the ASPIRE Filtering & Collection layer.

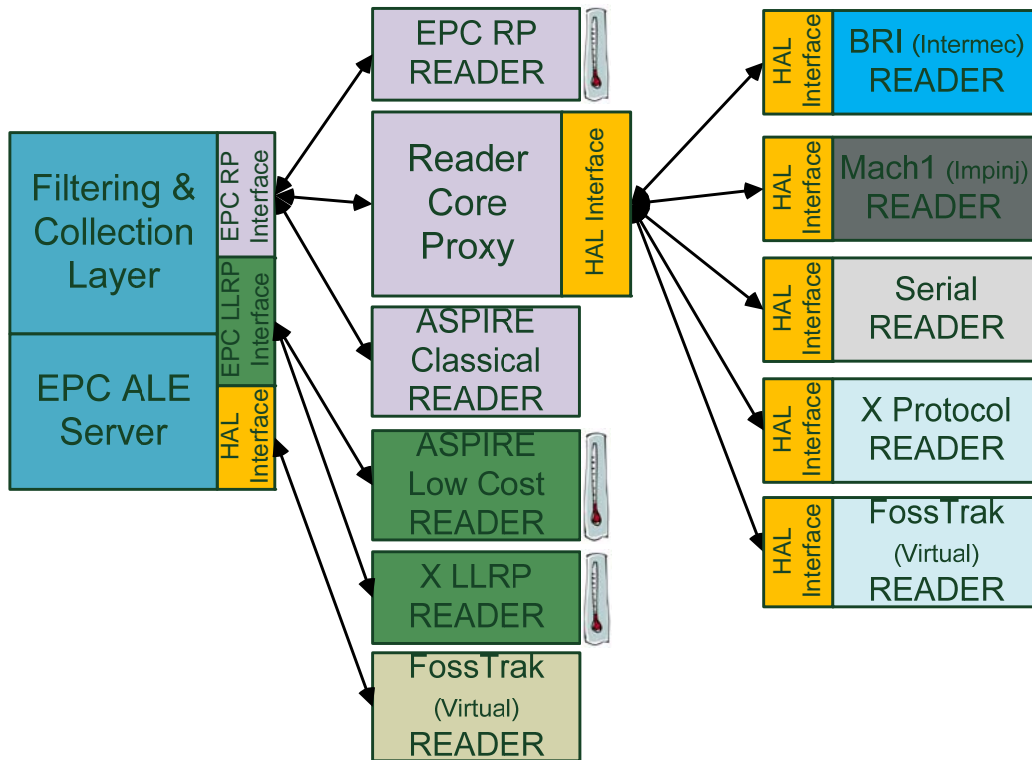


Figure 1: ASPIRE Interfaces

3.1 Low Level Reader Protocol Interface (LLRP)

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

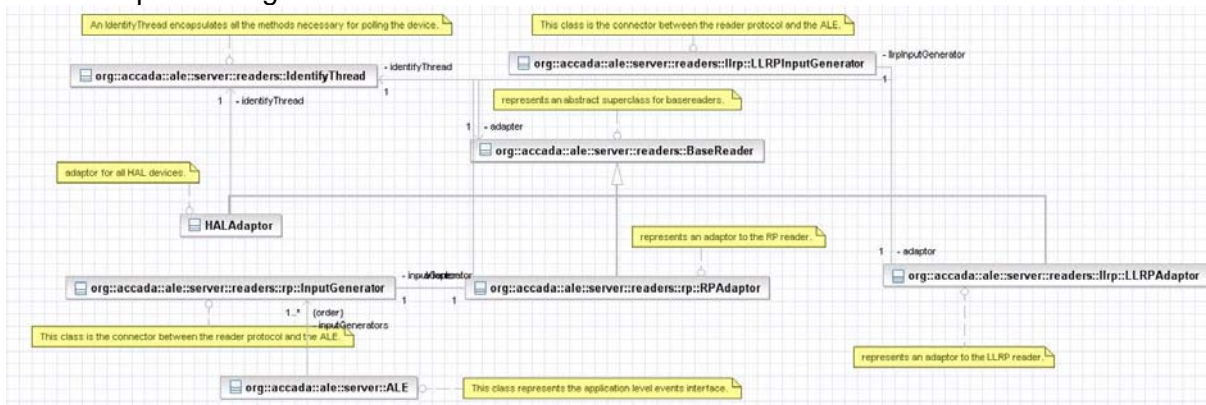


Figure 2: ASPIRE F&C Reader Interfaces

As shown in the UML diagram above (Figure 2) we are using the `LLRPAdaptor` class which extends the `BaseReader`. By overriding the following `BaseReader`'s methods:

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

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

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

- Setting readers configuration,
- Adding Readers Operation Specifications
- and finally enabling them.

`LLRPAdaptor` starts the `IdentifyThread` which is polling the LLRP Reader (i.e. depending on the defined boundary Specifications at the upstream F&C layer) by starting the Reader Operation Specifications every time an application subscribes to the F&C server and the defined F&C specification is using that LLRP reader for collecting reader data.

For the LLRP communication protocol we have used the LLRP toolkit library [9] which provides open source libraries in various languages to help reader and software vendors in building and parsing LLRP messages.

3.2 Reader Protocol Interface (RP)

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

As shown in the UML diagram above (Figure 2) we are using the `RPAdaptor` class which extends the `BaseReader`. By overriding the following `BaseReader`'s methods:

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

By using RP messages to implement the above primitives we achieve the communication between an RP Reader and the ASPIRE F&C Server.

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

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

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

For the RP protocol communication we are using the FossTrak Reader Proxy [1], which is a Java class library that supports the communication with a reader that implements the EPCglobal Reader Protocol Version 1.1.

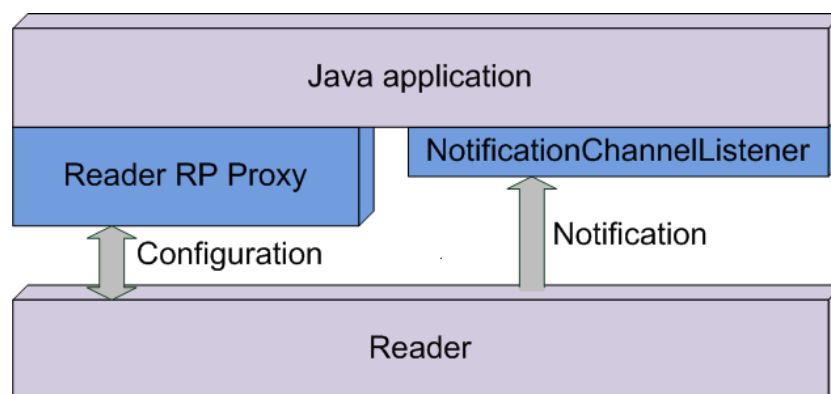


Figure 3: Use of FossTrak Reader Proxy [1]

The FossTrak Reader Proxy (which has been licensed by the FossTrak project) implemented elements of the protocol include:

- Transport Bindings: TCP and HTTP.
- Message Binding: XML and Text.
- Synchronous and Asynchronous Messaging (Notification Channels).
- Triggers.
- Data Selectors.

The reader as shown in Figure 3 above can be configured through a method call for each command or all the settings can be written into a configuration file which is processed by an automated configurator application.

3.3 Hardware Abstraction Layer Interface (HAL)

The third communication interface between ASPIRE architecture and an RFID reader is implemented indirectly or directly towards the F&C server by creating an appropriate class which implements Hardware Abstraction interface in case we want to use the Reader Core Proxy (indirect connection to the F&C server) or extend the class BaseReader in case we want to connect to the reader directly from the F&C server.

The objective of HAL is to define a hardware abstraction interface that is used to access RFID readers and implement it for various reader devices and reader simulators. HAL provides a common interface and wrappers implementing it to uniformly access the various RFID readers. Please note that HAL is needed for all readers that do not provide readily available support for the EPC-RP and EPC-LLRP protocols. Recently, we are witnessing the proliferation of readers that support these protocols. However, there are still several readers (including also legacy RFID interrogators) that do not support these protocols. The latter

require a specialized HAL implementation, in order to be used with the ASPIRE middleware. Based on the HAL abstraction new readers can easily be added as new modules to the ASPIRE project. For these new readers the controller must implement the `HardwareAbstraction` interface and communicate with the reader over the corresponding protocol.

At the moment, the following readers that are not Reader Protocol compliant, do already have a HAL implementation:

- Impinj Speedway.
- Intermec IF5.
- FEIG ELECTRONIC OBID i-scan ID ISC.MR100/101.
- FEIG ELECTRONIC OBID i-scan ID ISC.LRU1000.

The full list of readers supported by the ASPIRE project will be constantly updated and available in the ASPIRE Wiki.

In the scope of the HAL adaptation, ASPIRE is currently using the FossTrak simulator framework [1], in order to simulate demonstration scenarios in the absence of real readers. In this framework, there are several simulator controllers, each one implementing the `HardwareAbstraction` interface. This allows accessing the simulators in exactly the same way as HAL implementations for hardware readers. ASPIRE includes also its own simulator (namely fictive) as an OSGi bundle according to the ASPIRE architecture described in deliverable D2.3a.

3.3.1 Rationale behind using HAL

Each reader has to be accessed through its own proprietary protocol. This is very inflexible. If you access the reader directly and want to exchange it with another model, you have to adjust your application to control the reader through a different protocol. Using HAL we solve this problem by providing one single interface to access all the implemented readers. With this the code specific to the reader is moved from the application to the hardware abstraction layer. The application does not need to adjust to a specific reader but simply uses the `HardwareAbstraction` interface to access all readers. Overall, HAL ensures that applications remain independent of the underlying reader vendor: As soon as two or more readers provide equivalent functionalities, they are interchangeable (at least for these functionalities).

3.3.2 HAL Architecture

The following figure gives an overview of the HAL architecture devised according to [1].

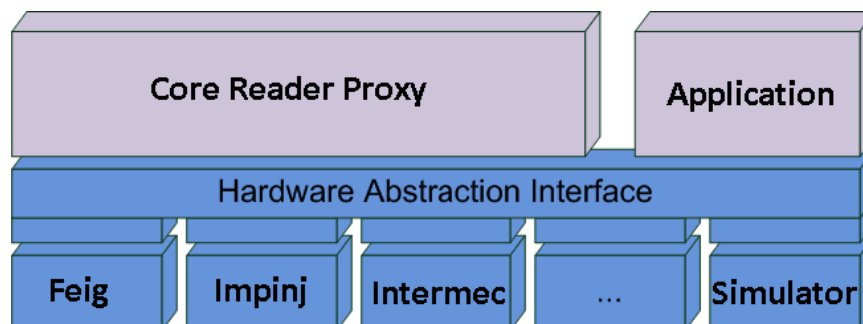


Figure 4: HAL Architectural Overview (according to [1])

The `HardwareAbstraction` (Figure 4) defines the interface between the HAL and the Reader Core 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 `HardwareAbstraction` 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.

The interface's methods are divided into the following groups:

- Identify, which reads tags in a synchronous ("push") fashion.
- Asynchronous Identify, , which reads tags in a asynchronous ("pull") fashion.
- Read and Write, enabling to data to be read from or written to a specified tag in range. Read and Write methods are used by the above Identify and Asynchronous Identify methods.
- Kill and WriteId, which enable: (a) killing the specified tags in range and (b) writing a new ID onto a tag independent of its position in memory, respectively.
- Controller Management, enabling acquisition and processing of management information concerning the reader. Moreover, it allows applications to control the reader via the HAL.
- Observation, which connects read tags with read points information.

3.3.3 Reader Core Proxy

In order to yield a non EPC-RP compliant reader from a non compliant reader, we have licensed and re-factored the FossTrak application called Reader Core. This application being used (as shown in Figure 4) as a mediate between an non-compliant reader and the F&C Reader Protocol Interface. By deploying the appropriate HAL module at the Reader Core we make the reader EPC-RP compliant. This means that the simulators and every reader with an implementation of the `HardwareAbstraction` interface can be used and controlled over the Reader Protocol. Also EPC Global Reader Management is implemented within the Reader Core so someone can monitor the operating status and health of the connected RFID Readers to it through SNMP messages.

The Reader Core Features are summarized below:

- Transport Binding: TCP and HTTP.
- Message Binding: XML and Text.
- Synchronous and Asynchronous Messaging (Notification Channels).
- Triggers.
- Data Selectors.
- SNMP Binding of Reader Management 1.0.

Applications for the Reader Core application include (see also [1], [10]):

- Turn a reader that does not implement the EPCglobal Reader Protocol itself into a compliant reader.
- Enable addition of new modules with an implementation of the `HardwareAbstraction` interface for other readers.
- Use an existing or write your own HAL wrapper for your reader and then leverage the filtering, event generation and messaging provided by the reader implementation.
- Simulate a single reader via a graphical user interface.
- Simulate a network of many readers using the simulation framework of the FossTrak HAL project.
- Embed the reader implementation into your reader.

3.3.3.1 Architectural Overview

The Reader Core uses the HAL as a backend and can be controlled and used like any other Reader Protocol compliant reader. That means it can be controlled using the Reader Proxy or any other application that supports the Reader Protocol. Additionally the Reader Core implements the SNMP part of the EPCglobal Reader Management so it can be monitored and managed with any management software that supports SNMP.

The Reader Proxy can be used by any Java application as a library to be able to easily control a reader over the EPCglobal Reader Protocol. This module shares code located in the Reader Core module. The proxy does not support Reader Management over SNMP.

3.3.3.1.1 Configuring the Reader

The Reader Core has the basic configuration values like every RFID reader that can be changed through an XML file placed inside the applications resource folder. Through this file one can set the:

- Information about the reader
 - Set the Reader EPC.
 - Set the Reader Name.
 - Set the Reader Manufacturer.
 - Set the Reader Manufacturer Description.
 - Set the Reader Model.
 - Set the Reader Role.
- Set More Information
 - Set the Max Source Number.
 - Set the Max Tag Selector Number.
 - Set the Max Trigger Number.
- Set All readers HAL's (`HardwareAbstractions`) that are used
- List of all The Sources.
- Set the IO trigger classes.
- Set the Information used for the reader management implementation:
 - Set the Description.
 - Set the Location Description.
 - Set the Contact.
 - Set the Serial Number.
 - Set the Mgmt Agent Type.
 - Set the Mgmt Agent Address.
 - Set the Mgmt Agent Port.
 - Set the MAC Address.
 - Set the Mgmt Simulator Start.
- Set the list of all alarm channels.
- Set the Information about the messaging:
 - Set if it supports TCP Server Connection.
 - Set the TCP Port.
 - Set if it supports HTTP Server Connection.
 - Set the HTTP Port.
 - Set the notification Listen Timeout.
- And finally can set the Information about the source defaults.

The properties `tcpServerConnection` and `tcpPort` specify if the command channel via TCP is active and which port the reader is listening for incoming TCP connections. Likewise, the properties `httpServerConnection` and `httpPort` define if the HTTP command channel is active and which port is used. The property `notificationListenTimeout` sets

the time in ms a notification connection waits in listen mode. The properties `threadPoolSize`, `startPattern` and `stopPattern` must not be changed to ensure correct operation of the reader. The information about the reader are properties describing the reader including the `epc`, `name` and `manufacturer` of the reader. The properties `maxSourceNumber`, `maxTagSelectorNumber` and `maxTriggerNumber` set a limit for the number of sources, tag selectors and triggers which a client can define. In the reader section the different instances of the HAL with their read points are given. The sources of the reader are specified by their name and the read points through which they acquire the RFID data. If the reader provides IO edge triggers or IO value triggers the classes that implement the functionalities to access the IO ports are specified by the two properties `IOEdgeTriggerPortManager` and `IOValueTriggerPortManager`. The remaining properties define the information used for the reader management. The properties `description`, `locationDescription`, `contact` and `serialNumber` specify additional information about the reader. The SNMP agent is specified by the properties `mgmtAgentType`, `mgmtAgentAddress`, `mgmtAgentPort`, `macAddress` and `mgmtSimulatorStart`. Alarm channels can be defined using the properties `alarmChannels`.

3.3.4 Various Vendors Reader Protocols

One can implement a HAL for practically any reader that exists so this way ASPIRE Middleware can support practically any reader. So far four readers already have a HAL implementation for their proprietary protocol:

- Intermec IF5 which is supporting the BRI protocol.
- Impinj Speedway which is supporting Mach1 and LLRP protocol (so it can be connected either directly to the F&C server or via the Reader Core proxy)
- FEIG ID ISC.LRU1000 (Ethernet, TPC/IP)
- FEIG ID ISC.MR101-A (RS232/485, COM)
- Simulator Controller (FossTrack [1]).

The implementations are tested on the reader models mentioned above but may work for other readers with the same interface and protocol.

3.4 ASPIRE Classical & Low-Cost reader

Although ASPIRE Low-Cost reader is still at the “design table” its main features, functionalities and capabilities have already been described but mentioning them is out of the scope of this document so only reader’s connection with the rest infrastructure will be specified.

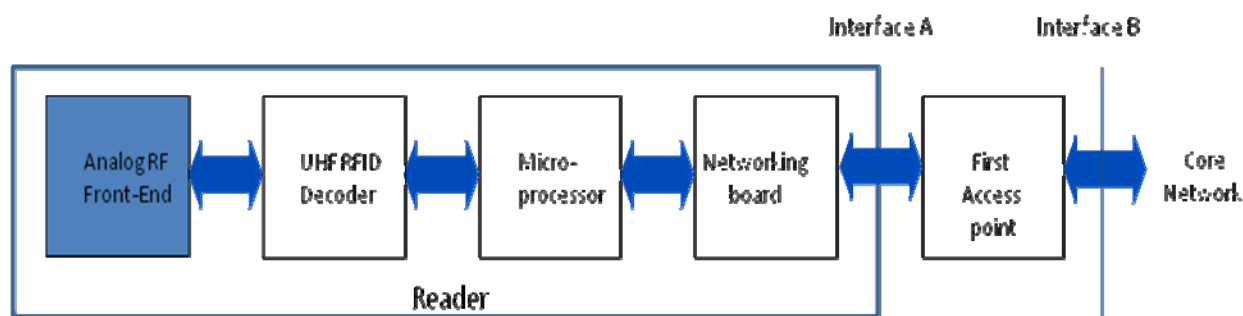


Figure 5: Structure of the low cost reader and the networking interfaces.

Two flavours of the ASPIRE reader will be created. One with lower capabilities the so called “Low Cost reader” that will use EPC LLRP connection interface and another called the “Classical Reader” that will use EPC RP connection Interface. RM (Reader management) and DCI will also be used for the middleware/reader interface.

So the most probable reader architecture to be deployed for the ASPIRE low cost/classical reader will be centralized with some variations according to the processing capabilities of the first access point in Figure 5. It is envisioned that this first access point will perform some of the functionalities of the controller device in the centralized architecture as shown in Figure 6. However, in the case of a mobile device with limited processing capabilities, it will be probably necessary to remove the functionalities of the controller device and use the second architecture also shown in Figure 6. Another difference between the architectures presented in Figure 6 with the architecture for the ASPIRE low cost reader in Figure 5 is that the link between the reader and the controller appliance or application server is not through a local area network (something that applies to the classical reader), but instead through a wireless personal area network based on Bluetooth technology (see Figure 6). Although it is expected that this difference will not implicate a radical change in system performance, some minor changes in the functionalities will be required.

So the low cost & classical reader/middleware interface will support the procedures of LLRP, RM, DCI, and RP, within the context of the EPC standards [2], [3], [4].

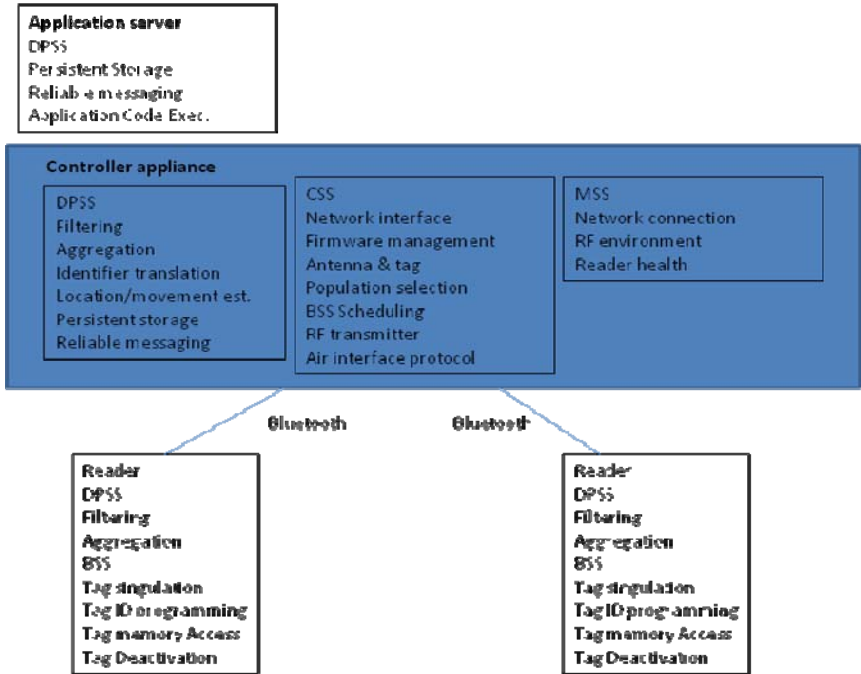


Figure 6: Centralized reader architecture for the ASPIRE low cost reader.

Section 4 - Tag Data Standards

4.1 EPC global Tag Data Standards

All the EPC global Tag Data standards [6] are summarized in the table below:

Identity Type	Tag Encodings	Header Value	Related GS1 Code
GID	GID-96	35	NA
SGTIN	SGTIN-96 SGTIN-198	30 36	GTIN (with added serial #)
SSCC	SSCC-96	31	SSCC
SGLN	SGLN-96 SGLN-195	32 39	GLN (with additional serial #)
GRAI	GRAI-96 GRAI-170	33 37	GRAI
GIAI	GIAI-96 GIAI-202	34 38	GIAI
DoD	DoD-96	2F	NA

4.1.1 GID

The General Identifier (GID-96) is defined as one general identity type and is composed of three fields:

- the General Manager Number is a unique number (the uniqueness is ensured by EPCglobal) assigned by EPCglobal to an organizational entity, which is responsible for maintaining the numbers in subsequent fields – Object Class and Serial Number;
- the Object Class must be unique within each General Manager Number domain and identify a class or “type” of thing;
- the Serial Number is unique for every instance within each object class.

	Header	General Manager Number	Object Class	Serial Number
GID-96	8 bits	28 bits	24 bits	36 bits

4.1.2 SGTIN

The Serialized Global Trade Item Number (SGTIN) is based on the GS1 GTIN code augmented with a serial number. So it identifies uniquely a single physical object. It is composed of five fields:

- the Filter Value is used for fast filtering and pre-selection of basic logistics types, it is not a part of the SGTIN pure identity;
- the Partition defines the size of the Company Prefix and the Item Reference;
- the Company Prefix identifies a managing entity, and is assigned by GS1;

- the Item Reference identifies a particular object class, and is assigned by the managing entity;
- the Serial Number identifies an individual object, and is assigned by the managing entity.

	Header	Filter Value	Partition	Company Prefix	Item Reference	Serial Number
SGTIN-96	8 bits	3 bits	3 bits	20-40 bits	24-4 bits	38 bits
SGTIN-198	8 bits	3 bits	3 bits	20-40 bits	24-4 bits	140 bits

4.1.3 SSCC

The Serial Shipping Container Code (SSCC) is composed of five fields:

- the Filter Value is used for fast filtering and pre-selection of basic logistics types, it is not a part of the SSCC or EPC identifier;
- the Partition defines the size of the Company Prefix and the Serial Reference;
- the Company Prefix identifies a managing entity, and is assigned by GS1;
- the Serial Reference identifies a specific shipping unit, and is assigned by the managing entity;
- the Unallocated field is not used in the 1.4 specification of EPC tag data standard and must contain zero.

	Header	Filter Value	Partition	Company Prefix	Serial Reference	Unallocated
SSCC-96	8 bits	3 bits	3 bits	20-40 bits	38-18 bits	24 bits

4.1.4 SGLN

The Serialized Global Location Number (SGLN) is composed of five fields:

- the Filter Value is used for fast filtering and pre-selection of basic location types, it is not a part of the GLN or EPC identifier;
- the Partition defines the size of the Company Prefix and the Location Reference;
- the Company Prefix identifies a managing entity, and is assigned by GS1;
- the Location Reference identifies an aggregate or a specific physical location, and is assigned by the managing entity;
- the Extension Component identifies an individual unique location and is assigned by the managing entity (optional).

	Header	Filter Value	Partition	Company Prefix	Location Reference	Extension Component
SGLN-96	8 bits	3 bits	3 bits	20-40 bits	21-1 bits	41 bits (if not used, this field must contain zero)
SGLN-195	8 bits	3 bits	3 bitd	20-40 bits	21-1 bits	140 bits (if not used, this field must contain 0110000 followed by 133 binary 0 bits)

4.1.5 GRAI

The Global Returnable Asset Identifier (GRAI) is composed of five fields:

- the Filter Value is used for fast filtering and pre-selection of basic asset types, it is not a part of the GRAI or EPC identifier;
- the Partition defines the size of the Company Prefix and the Asset Type;
- the Company Prefix identifies a managing entity, and is assigned by GS1;
- the Asset Type identifies a particular class of asset, and is assigned by the managing entity;
- the Serial Number identifies an individual object and is assigned by the managing entity.

	Header	Filter Value	Partition	Company Prefix	Item Reference	Serial Number
GRAI-96	8 bits	3 bits	3 bits	20-40 bits	24-4 bits	38 bits
GRAI-170	8 bits	3 bits	3 bits	20-40 bits	24-4 bits	112 bits

4.1.6 GIAI

The Global Individual Asset Identifier (GIAI) is composed of four fields:

- the Filter Value is used for fast filtering and pre-selection of basic asset types, it is not a part of the GIAI or EPC identifier;
- the Partition defines the size of the Company Prefix and the Individual Asset Reference;
- the Company Prefix identifies a managing entity, and is assigned by GS1;
- the Individual Asset Reference identifies a specific asset and is assigned by the managing entity.

	Header	Filter Value	Partition	Company Prefix	Individual Asset Reference
GIAI-96	8 bits	3 bits	3 bits	20-40 bits	62-42 bits
GIAI-202	8 bits	3 bits	3 bits	20-40 bits	168-148 bits

4.1.7 DoD

The DoD identify type is defined by the United States Department of Defense (Consult US Department of Defense doc for details)

	Header	Filter Value	Government Managed Identifier	Serial Number
DoD-96	8 bits	4 bits	48 bits	36 bits

4.2 ISO Tag Data Standards

4.2.1 ISO 15961

ISO 15961 focuses on the interface between the application and the data protocol processor, and includes the specification of the transfer syntax and definition of application commands and responses. It allows data and commands to be specified in a standardized way, independent of the particular air interface of ISO 18000.

4.2.2 ISO 15962

ISO 15962 focuses on encoding the transfer syntax, as defined in ISO 15961 according to the application commands defined in that international standard. The encoding is in a logical memory as a software analogue of the physical memory of the RF tag being addressed by the interrogator.

4.2.3 ISO 15963

ISO 15963 describes numbering systems that are available for the identification of RF tags. A unique ID is required as part of the write operation to RFID tags. The unique ID guarantees that the information written to a tag is unambiguously written to the correct data carrier (tag). A unique ID is also required in read situations where the contents of the tag are tied to a specific item and that item needs to be unambiguously identified.

4.2.4 ISO 15693

ISO 15693 focuses on contactless integrated circuit(s) cards (Vicinity cards). The vicinity cards are uniquely identified by a 64 bits UID. This UID (Unique Identifier) shall be set permanently by the manufacturer.

The UID is composed of three fields :

- the 8 most significant bits shall be 'E0';
- the IC manufacturer code, on 8 bits according to ISO/IEC 7816-6/AM1;
- a unique serial number on 48 bits assigned by the IC manufacturer.

4.2.5 ISO 14443

ISO 14443 describes the contactless integrated circuit(s) cards (Proximity cards). There are three sizes of UID for proximity cards, called single, double and triple.

Single size UIDs (4 bytes):

uid0	Description
'08'	uid1 to uid3 is a random number which is dynamically generated
'x0' – 'x7' 'x9' – 'xE'	Proprietary fixed number
'18' – 'F8' 'xF'	Reserved for futur use

(uid0 represents the first byte of the UID)

Double and triple size UIDs (respectively 7 and 10 bytes):

uid0	Description
Manufacturer ID according to ISO/IEC 7816-6/AM1	Each manufacturer is responsible for the uniqueness of the value of the other byte of the unique number.

4.3 GS1 Bar Code Data Standards

The symbology identifiers is generated by the decoder after decoding and transmitted as a preamble to the data message, but is not embedded in the bar code symbol.

Symbology	Symbology Format	Content
-----------	------------------	---------

Identifier		
] E 0	EAN-13, UPC-A or UPC-E	13 digits
] E 1	Two-digit Add-On Symbol	2 digits
] E 2	Five-digit Add-On Symbol	5 digits
] E 3	EAN-13, UPC-A or UPC-E with Add-On Symbol	15 or 18 digits
] E 4	EAN-8	8 digits
] I 1	ITF-14	14 digits
] C 1	GS1-128	Standard Application Identifiers Elements Strings
] e 0	GS1 DataBar	Standard Application Identifiers Elements Strings
] d 2	Data Matrix	Standard Application Identifiers Elements Strings

Application Identifiers	Data Content
00	SSCC
01	GTIN
02	GTIN
414	GLN
8003	GRAI
8004	GIAI
21	Serial Number
254	GLN Extension Component

The EPC SGTIN standard may be considered equivalent to a GTIN standard and a serial number (application identifiers 01 and 21). In EPC SGTIN standard, the serial number is mandatory.

The EPC SGLN standard may be equivalent to a GLN standard with an optional GLN Extension Component (414 and 254). In EPC SGLN standard, the extension component is optional.

Section 5 – ASPIRE Tag Data Translation Implementation

5.1 From Barcode to EPC TDS

GS1 DataBar expanded versions, DataMatrix and GS1-128 bar code types can carry all EPC tag data standards, and can be easily embedded in the EPC tag data translation process.

Exemple of EPC string format	Bar code symbology
gtin:00037000302414;serial:1041970	(01)00037000302414(21)1041970
sscc:000370003024147856	(00)000370003024147856
gln:0003700030241;serial:1041970	(414)0003700030241(254)1041970
grai:00037000302414274877906943	(8003)00037000302414274877906943
giai:00370003024149267890123	(8004)00370003024149267890123

Other bar code types can only encode a GTIN without serial number. For homogeneity concerns, this kind of code will use the same input as the others by setting the serial number to 0 (see section 5.4 for details).

5.2 From ISO to EPC TDS

The ISO normalization describes several kinds of tags different from EPC, specifying the data to embed but by no means it cut up each field. It means that each user may program by itself the size of fields, which leads to the impossibility to identify them clearly. When using such tags, the use of a dedicated IS is mandatory but this latter would be different from an EPC IS since it is personalized by the user. Therefore, the TDT can only identify those tags and redirect them towards their dedicated IS.

5.3 From Various EPC Formats to other EPC Formats

The aim of the EPC tag data translation is to convert one representation of an EPC code into another (binary, tag-encoding URI, pure-identity URI, legacy or ONS hostname).

As shown in Figure 10 and Figure 11, every EPC tag can be re-written in an EPC URI which will redirect it towards the corresponding EPCIS, under condition that the identifier is registered in the EPC IS. In the same way, from the EPC IS through the TDT, it will be possible to write information coming from the application into tags that allow writing actions (RFID).

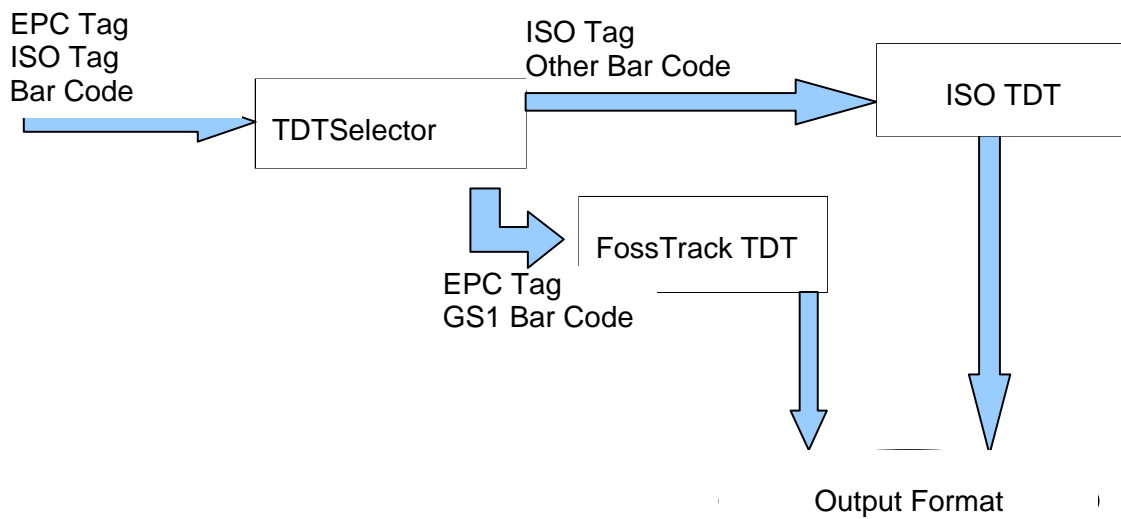


Figure 9: Aspire Tag Data Translation Process

5.5 Tag Data Translation Roadmap

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

Tag Data Standards	Supported
Bar Code Tags (GS1 System)	Yes
EAN/UPC	Yes
ITF-14	Yes
GS1 DataMatrix	Yes
GS1 DAtaBar	Yes
GS1-128	Yes
EPC Global Tags	Yes
SSCC	Yes
GTIN	Yes
GTIN	Yes
GLN	Yes
GRAI	Yes
GIAI	Yes
GLN	Yes
ISO Tags	To be supported
14443,	To be supported
15693	To be supported
15962	To be supported
uCode Tags	To be supported

Section 6 - Conclusions

This deliverable has described the mechanisms that enable the ASPIRE middleware platform to be independent of RFID reader and tag vendors. Reader vendor independence is ensured through the use of standard messaging protocols for accessing the reader capabilities. For several readers, i.e. those that are not compliant to the used standards, developers have to provide an implementation of a set of interfaces to the low level capabilities of the reader. These interfaces form a hardware abstraction layer (HAL) for the ASPIRE middleware. The ASPIRE consortium has already provided HAL implementations for various vendors, which validates the concept of reader virtualization. As a complement to reader virtualization, the ASPIRE middleware tag independence relies on the implementation of several mapping (i.e. tag translation mechanisms) enabling different tag types and bar codes to be used within the ASPIRE middleware. Community developers will be encouraged to contribute to additional HAL implementations and tag mappings based on their particular projects and needs.

Independence from particular vendors and hardware is among the key requirements for Small Medium Enterprise (SMEs), which want to avoid vendor lock-ins. Moreover, SMEs need to be able to combine RFID with legacy bar-code solutions. These requirements are fully satisfied by the ASPIRE middleware, as illustrated in this deliverable.

The deliverable includes also a prototype implementation of the implemented features. This prototype implementation will become part of the “AspireRfid” project (<http://forge.objectweb.org/projects/aspire/>).

Section 7 - Acronyms

ASPIRE: Advanced Sensors and lightweight Programmable middleware for Innovative Rfid Enterprise applications

ALE : Application Level Events

DCI : Discovery Configuration and Initialization

EPC : Electronic Product Code

EPCIS : EPC Information Services

HAL : Hardware Abstraction Layer

RFID : Radio Frequency Identification

RM : Reader Management

RP : Reader Protocol

LLRP : Low Level Reader Protocol

SNMP : Simple Network Management Protocol

GTIN : Global Trade Item Number

SSCC : Serial Shipping Container Code

GLN : Global Location Number

GRAI : Global Returnable Asset Identifier

GIAI : Global Individual Asset Identifier

GID : General Identifier

GDTI : Global Document Type Identifier

GSRN : Global Service Relation Number

List of Figures

Figure 1: ASPIRE Interfaces	9
Figure 2: ASPIRE F&C Reader Interfaces	9
Figure 3: Use of FossTrak Reader Proxy [1]	11
Figure 4: HAL Architectural Overview (according to [1])	12
Figure 5: Structure of the low cost reader and the networking interfaces.	15
Figure 6: Centralized reader architecture for the ASPIRE low cost reader.	16
Figure 7: Tag Data Translation – Concept [7].	23
Figure 8: Tag Data Translation process with examples of different representations [7].	23

Section 8 - 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] LLRP Toolkit, <http://www.llrp.org/>
- [9] C.Floerkemeier, C. Roduner, and M. Lampe, RFID Application Development With the Accada Middleware Platform, IEEE Systems Journal, Vol. 1, No. 2, December 2007.
- [10] C. Floerkemeier and S. Sarma, An Overview of RFID System Interfaces and Reader Protocols, 2008 IEEE International Conference on RFID, The Venetian, Las Vegas, Nevada, USA, April 16-17, 2008.
- [11] Michel Cezon, Guillaume Vaudaux-Ruth, Luc Laurens, John Soldatos, et. al., "Review of State-of-the-Art Middleware", ASPIRE Project Public Deliverable D2.1, June 2008, available at: <http://www.fp7-aspire.eu>
- [12] J. Soldatos, P. Dimitropoulos, A. Anagnostopoulos and S.Michalakos 'Lightweight Programmable Middleware for Cost Effective RFID Application Development', in the Proc. of the 15th IST Mobile and Wireless Communication Summit, Myconos Greece, June 4-8, 2006 (invited paper).
- [13] Christian Floerkemeier and Matthias Lampe, "RFID middleware design: addressing application requirements and RFID constraints" in ACM International Conference Proceeding Series; Vol. 121, Proceedings of the 2005 Joint conference on "Smart objects and ambient intelligence: innovative context-aware services: usages and technologies table of contents", pp. 219 – 224, 2005
- [14] Nikos Zarokostas, Panagiotis Dimitropoulos and John Soldatos, "RFID Middleware Design for enhancing traceability in the Supply Chain Management", in the Proc. of the 18th IEEE Personal Indoor and Mobile Radio Communications, Athens, Greece, September 3-7, 2007.
- [15] Shawn R. Jeffery, Michael J. Franklin, and Minos Garofalakis. "An Adaptive RFID Middleware for Supporting Metaphysical Data Independence", The VLDB Journal, Vol. 17, No. 2, March 2008 ("Best of VLDB'2006" Special Issue).
- [16] The ASPIRE FP7 Project, <http://www.fp7-aspire.eu>
- [17] The AspireRfid project, <http://forge.objectweb.org/projects/aspire/> (forge), <http://wiki.aspire.objectweb.org/xwiki/bin/view/Main/WebHome> (wiki)
- [18] EPC DCI (Discovery, Configuration & Initialization Standard for Reader Operations) standard (in development), available at: <http://www.epcglobalinc.org/standards/dci>